# Ein archäologisches Bauwerksinformationssystem

Magisterarbeit an der
Philosophischen Fakultät der
Universität zu Köln
vorgelegt von Florian D. W. Willems

# Inhaltsverzeichnis

# Kapitel 1

# Vorwort

Diese Arbeit behandelt den Sinn und die Art der Realisierung der Verbindung eines rekonstruierten, archäologischen 3D-Modells eines nur noch in Ruinen existenten Gebäudes mit einer bestehenden Forschungsdatenbank für archäologische Objekte. Das gewählte Beispiel ist die Basilica Aemilia, eines der größten Gebäude am Forum Romanum, dem hauptstädtischen Mittelpunkt des Imperium Romanum. Die bestehende Forschungsdatenbank ist Arachne, die Bilddatenbank des Forschungsarchivs für Antike Plastik des Archäologischen Instituts der Universität zu Köln.
Die Arbeit[1] beinhaltet einen archäologischen Überblick über die Situation der Basilica, die zugrunde liegenden informationstechnischen Paradigmen und schließlich die Dokumentation der Realisierung eines Design- und Funktionsprototypen.

Der hier gewählte Ansatz behandelt ein methodisches Problem der klassischen Archäologie. Häufig sind es Bauteile, auf Basis derer ein Gebäude von Archäologen rekonstruiert wird. Dies beinhaltet maßgeblich die Verortung der Bauteilgruppen an bestimmten Plätzen im Bauwerk. Bei sehr schlecht erhaltenen Bauwerken ist es meist so, daß es mehrere Vorschläge für mögliche Rekonstruktionen gibt. Dieses Projekt soll es ermöglichen, die Platzierung dieser Bauteilgruppen im Gebäude zu visualisieren und bei mehreren Rekonstruktionsvorschlägen – und für diese Rekonstruktionsvorschläge vorhandenen Modellen – die unterschiedliche Platzierung der Bauteilgruppen

[1]Ein Teil der Arbeit entstand innerhalb eines Projekts des Deutschen Archäologischen Instituts zur Bauneuaufnahme und retrospektiven Sichtung der bisherigen Bauaufnahmen unter Leitung von K. S. Freyberger und H. v. Hesberg, denen ich auch für die Möglichkeit, Einsicht in die neuesten Forschungsergebnisse zu erhalten, danken möchte. Bedanken möchte ich mich zudem bei R. Förtsch für Möglichkeiten und Geduld und bei M. Gärtner und R. Grabowski für Kommunikation und Beistand.

deutlich zu machen.
Technisch gliedert sich der praktische Teil in vier Elemente:

- die bestehende Forschungsdatenbank, worin die Bauglieder mit ihren archäologischen Daten gespeichert sind,
- den Brower, der die Benutzeroberfläche für die Datenbank darstellt,
- ein dreidimensionales Interface für die Visualisierung des Modells,
- ein Proxyserver, der die Kommunikation zwischen den drei vorgenannten Elementen moderiert.

Es soll gezeigt werden, daß mit minimalen Veränderungen an der bestehenden Datenbank eine solche Lösung so implementiert werden kann, daß sie dem methodischen Anspruch der Visualisierung einer oder mehrerer Rekonstruktionen gerecht wird. Ein weiterer Anspruch ist die einfache Erweiterbarkeit, um zukünftige Rekonstruktionsprojekte und eine sehr leicht zu handhabende Bedienoberfläche.

# Kapitel 2

# Archäologische Grundlagen

In diesem Kapitel soll dargelegt werden, welche archäologischen Bedingungen und Methoden evaluiert und bedacht werden müssen, um zu einer sinnvollen Lösung zur Verbindung von archäologischem Datenmaterial, sprich eines Rekonstruktionsversuches in 3-D, und einer bestehenden Forschungsdatenbank zu kommen.
Zusätzlich werden noch die Gründe beleuchtet, wegen derer die Wahl des Beispielobjekts auf die Basilica Aemilia fiel.

## 2.1 Archäologische Rekonstruktionen

Die Archäologie beschäftigt sich mit den materiellen Hinterlassenschaften vergangener Kulturen und dem Versuch einer Rekonstruktion der Lebensumstände der Menschen, die in ihnen lebten. Ein weiterer wichtiger Aspekt – besonders für die klassische Archäologie – ist die Kunstgeschichte der untersuchten Kulturen, im Fall der klassischen Archäologie der Kulturen der Griechen, Römer und ihrer Nachbarn.

Das Material mit den fast besten Erhaltungsbedingungen ist auch jenes, aus dem die größten Monumente gebaut wurden: Stein. Seine ubiquitäre Verfügbarkeit, die verhältnismäßig einfachen Bearbeitungsmöglichkeiten und die Möglichkeit, mehrere Werkblöcke zu einem wesentlich größeren Ganzen zusammensetzen zu können, führten alle dazu, daß ein Großteil der mehr oder weniger erhaltenen Monumente der Kulturen des Mitttelmeerraumes aus Stein sind. Die letztgenannte Eigenschaft der Zusammensetzbarkeit führte aber auch dazu, daß bereits bearbeitete und verbaute Steine in Zeiten der Not oder des Niedergangs aus ihrem ehemaligen Gefüge herausgebrochen und in Zweitkontexten wiederverwendet wurden. Andere Bauten wurden bei Na-

turkatastrophen oder durch Kriege zerstört.
Dieser Umstand führt dazu, daß über die ursprüngliche Gestalt eines Gebäudes häufig nichts bekannt ist außer seinem Grundriß und einigen dem Gebäude zuzuordnenden Bauteilen. Einer der wichtigen Teilbereiche der Archäologie widmet sich dem Versuch, über Analysen und Interpretationen die ursprüngliche Gestalt eines Gebäudes zu rekonstruieren.

### 2.1.1 Material

Das zusammenhängendste, was von einem Gebäude übrig bleibt, sind meist die Grundmauern. Häufig sind auch noch einige aufgehende Mauern erhalten, an denen sich die Mauertechnik und im Idealfall die Höhe der Geschosse ablesen läßt. Sehr oft findet man in der Umgebung des Gebäudes Bauteile wie Architrave oder Kapitelle, die wegen ihres Stils und ihrer Größe dem Gebäude zugeordnet werden können. Auch diese Bauornamentik hilft bei der Rekonstruktion der ursprünglichen Gestalt: Bei Säulen sind beispielsweise die Verhältnisse zwischen Durchmesser und Höhe für die einzelnen Bauordnungen weitestgehend kanonisch. Auch Architravhöhen und die Größe von Dübellöchern kann einiges über die Dimensionierung eines Gebäudes sowie über nicht mehr erhaltene Bauteile aussagen, da in bestimmten Zeiten und in bestimmten Regionen die Abfolge von Elementen und deren Aussehen häufig einem festen Schema folgten.

### 2.1.2 Paradigmen

Eine etwas flapsig formulierte Grundannahme der Archäologie, die fast jeder Student im Grundstudium als bewußt dehnbaren Leitsatz mit auf den Weg gegeben bekommt, lautet “Einmal ist keinmal, zweimal ist immer”. Anders formuliert: Wenn etwas nur ein einziges Mal auftaucht, ist es gut möglich, daß es sich um eine singuläre Erscheinung handelt, wenn man jedoch mehr als eine Evidenz hat, kann man es als Arbeitshypothese durchaus auch auf andere Kontexte übertragen und es dort auf seine Beweisbarkeit überprüfen. Bezogen auf die Rekonstruktion eines schlecht erhaltenen Gebäudes, und auch noch eines Gebäudes mit einer großen Bedeutung, heißt das, daß gewisse Gegebenheiten in ähnlichen Kontexten, anderen Basiliken zum Beispiel, mit relativ großer Wahrscheinlichkeit bei gewissen Indizien auch auf das betreffende Gebäude übertragbar sind. Wenn es beispielsweise Aufleger auf den Architraven gibt, kann man aus dem Vergleich mit Architraven besser erhaltener Gebäude und ein wenig statischer Überlegung extrapolieren, ob nun ein Tonnengewölbe existierte oder nicht.

Ein weiterer Punkt ist ein in der gesamten Antike in offiziellen Bauten zu beobachtender Wiederholungsfaktor. Es ist mehr als unwahrscheinlich, daß die Säule neben einer anderen grundlegend anders aussieht, solange sie sich im selben Funktionskontext befindet. Wenn man nun ein dreischiffiges Gebäude hat, und Kapitelle zweier Serien, so ist anzunehmen, daß die eine Serie in das Mittelschiff, die andere in die Seitenschiffe zu rekonstruieren ist. Die Statik eines Gebäudes ist natürlich auch immer in die Überlegungen einzubeziehen: Wenn das rekonstruierte Dach partout nicht paßt, ist es wohl falsch. Wenn die rekonstruierten Tonnengewölbe keine Möglichkeit zur lateralen Kraftableitung haben, wird das wohl falsch sein. Der Beispiele gäbe es noch viele, die grundlegenden Gesetze der Physik sollten jedoch auch für Rekonstruktionen nicht ignoriert werden.

### 2.1.3 Methoden

Bauwerksrekonstruktionen in der klassischen Archäologie folgen im Idealfall dem folgenden Schema:

- Formulierung der Fragestellung
- Sichtung der bisherigen archäologischen Publikationen
- Sichtung der antiken Quellen
- Sichtung der sonstigen Quellen, zum Beispiel Renaissance-Zeichnungen
- Bauaufnahme vor Ort
- Aufnahme der erhaltenen Bauornamentik
- Erstellung möglicher Rekonstruktionen
- Quellenkritik

Die Formulierung der Fragestellung beeinflußt auch immer die Sichtweise auf die weiteren Glieder des Prozesses: Will der Wissenschaftler primär den Zweck eines Gebäudes kennenlernen? Die ursprüngliche Gestalt? Je nach Ansatz wird mehr oder weniger Gewicht auf antike Quellen gelegt, wird die Bauaufnahme maßgeblich auf die architektonischen Reste oder auf die Bauausstattung beschränkt. Auch gibt es große Projekte, die sich mit allen Aspekten des Gebäudes auseinandersetzen.

Sämtliche literarischen Quellen, seien es historische oder rezente Publikationen, müssen quellenkritisch auf ihren faktischen Gehalt hin überprüft

werden. Korrespondiert die Aussage des antiken Autors mit den baulichen Resten, die sich in situ befinden? Ist die Aussage des Archäologen, der sich dieses Gebäudes schon einmal angenommen hat, überprüfbar?

Die Bauaufnahme vor Ort wird meist von Architekten oder Archäologen mit dieser Zusatzqualifikation übernommen. Bestenfalls findet dies bereits elektronisch statt, so daß die steingenauen Pläne abstrahiert und zu einem dreidimensionalen Gebäude erweitert werden können. Wichtig sind hierbei vor allem die Bauphasen: Welche Mauer, welcher Boden gehört zu der Phase, die untersucht werden soll? Stratigraphien müssen erstellt werden, wichtige Details, die in einem Aspekt des Gebäudes vielleicht besser erhalten sind, müssen gefunden und auf das Gesamtgebäude extrapoliert werden.

Anschließend muß geklärt werden, welche Teile der Bauornamentik wirklich zu diesem Gebäude gehören, und wenn sie dazu gehören, aus welcher Phase sie stammen. Es folgt dann eine Serialisierung der Bauornamentik, um qualitative und quantitative Unterschiede festzustellen, und auf Basis dessen vielleicht zu einem Ergebnis über die Höhe und Anzahl der Stockwerke zu kommen.

### 2.1.4 Ziele

Das abstrakte, primäre Ziel einer archäologischen Gebäuderekonstruktion ist in erster Linie das gleiche wie in der gesamten Archäologie: eine möglichst genaue Kenntnis über die materiellen Hinterlassenschaften einer vergangenen Kultur zu erlangen und auf dieser Basis Aussagen über die Kultur und Geistesgeschichte treffen zu können. Ein prächtig mit Marmor verkleideter Bau hat eine andere Aussage als ein verputztes Backsteingebäude, wobei aber gegebenenfalls das weniger prächtige Gebäude mehr über das Selbstverständnis eines Herrschenden aussagen kann als hohe Marmorsäulen. Die Archäologie versucht, einen möglichst holistischen Eindruck des Erscheinungsbildes einer Kultur zu vermitteln, und da sind die materiellen Quellen das, was den technischen und visuellen Eindruck einer Kultur maßgeblich beeinflußt.
So hat eine Gebäuderekonstruktion folgende Hauptziele:

- einen möglichst vollständigen Eindruck der visuellen Erscheinung eines Gebäudes zu schaffen,
- die Funktion eines Bauwerks an seiner Gestalt analysieren zu können,
- daraus ableitend Aussagen über wirtschaftlichen Aufwand, soziale und

politische Bedeutung, Nutzungsdauer und -veränderungen und generelle Bedeutung treffen zu können.

All diese Ziele werden bei der Bauneuaufnahme der Basilica Aemilia betrachtet, was dieses Gebäude für eine Verbindung zwischen möglichen Rekonstruktionen und deren in einer Datenbank gespeicherten Grundlagen prädestiniert.

## 2.2 Die Basilica Aemilia

Die Basilica Aemilia, oder deren Ruinen, befindet[1] sich an der nordöstlichen Längsseite des Forum Romanum, zwischen der Curia und dem Tempel des Antoninus und der Faustina, und war nach Plinius eines der schönsten Bauwerke der römischen Antike[2]. Im Folgenden soll die Geschichte der Basilica kurz skizziert werden.

### 2.2.1 Geschichte der Basilica Aemilia

Abgesehen von Vorgängerbauten am selben Platz, die aber nicht Teil dieses kurzen Überblicks sein sollen, gliedert sich die (Bau-)Geschichte der Basilica Aemilia in drei Phasen, die weitgehend auch mit den groben Phasen der römischen Zeitgeschichte übereinstimmen.

#### Die republikanische Zeit

Greifbar wird die Basilica Aemilia erstmals im Jahre 179 v. Chr., als die Censoren dieses Jahres, M. Fulvius Nobilior und M. Aemilius Laepidus, die Steuerabgaben eines ganzen Jahres für die Durchführung öffentlicher Bauvorhaben zur Verfügung hatten[Kle93, Gro10]. Livius berichtet, daß es einige Bauvorhaben gab, die beide zusammen tätigten, aber daß auch jeder einzeln einige Bauten errichten ließ – so Fulvius eine Basilica "hinter den neuen Wechselstuben"[3], die heute gemeinhin als die sichbaren Ruinen der später so genannten Basilica Aemilia gelten.
Die nächste Kunde über dieses Gebäude stammt aus dem Jahr 159 v. Chr. und berichtet vom Einbau einer Wasseruhr in das Gebäude[Kol95, Seite 208].

Zwischen 80 und 78 v. Chr., so berichtet Plinius[4], ließ M. Aemilius Laepidus, ein Nachfahre des Vorgenannten, Schilde an der Basilica Aemilia an-

[1] 41 53' 33,32" N, 12 29' 09,90" O
[2] Plin. nat. 36, 102
[3] Liv. 40, 51
[4] Plin. nat. 35, 4

bringen, die Bildnisse seiner Vorfahren zeigten. Ein Teil der Forscher nimmt dieses Ereignis als den Beginn einer neuen Bauphase an, da M. Aemilius Lepidus, später Triumvir, einen Denar[Jr.79] herausgab, auf dem ein einstöckiges Gebäude mit einer Säulenstellung im Erdgeschoss und Obergeschoss zu sehen ist. Diese Darstellung wird wegen der Schilde zwischen den beiden Geschossen und der "aemilianischen Vestalin" auf der Vorderseite als Basilica Aemilia identifiziert, auch, da die Schilde bei Plinius genannt werden.
Die Datierung dieses Denars ist allerdings eine weitere Unklarheit in der Geschichte der Basilica Aemilia, da sich Angaben zwischen 66 und 54 v. Chr. in der Literatur finden, wobei eine Emission zwischen 61 und 58 v. Chr. als wahrscheinlich gilt. Eine weitere Frage wirft sich auf: Markierte die Emission des Denars nun das Ende einer 78 v. Chr. begonnenen Bauphase? Plinius spricht nicht von einer Restaurierung, sondern nur von der Anbringung der Schilde, und die Denare dieser Zeit weisen oft familiäre Themen ohne Bezug zu aktuellen Ereignissen auf.
Diese Probleme bleiben im Raum stehen, und nur weitergehende archäologische Forschungsarbeit wird vielleicht in der Lage sein, die tatsächlichen Ereignisse zu skizzieren.

In der Mitte des 1. Jahrhunderts v. Chr. läßt sich allerdings eine sichere Neubauphase belegen, da diese von gleich drei literarischen Quellen[5] erwähnt wird. L. Aemilius Paullus hatte – gemeinhin datiert in den Zeitraum von 55 bis 34 v. Chr. – diesen Neubau zu verantworten. Die Quellenlage hier ist aber äußerst verworren, so daß durch das Mehr an Quellen kein Zugewinn an sicheren Ereignissen verzeichnet werden kann.
Die älteste der Quellen ist der Cicerobrief, auf den 1. oder 2. Juli 54 v. Chr. datiert, der beschreibt, daß Paullus eine Basilica auf dem Forum unter Verwendung der alten Säulen bereits fertiggestellt habe. Plutarch und Appian hingegen berichten von einem Bestechungsgeld von 1500 Talenten, die Paullus 51/50 v. Chr. von Caesar erhalten habe und von dem er den Bau der Basilica, die seinen Namen tragen sollte, finanziert habe. Dieses Problem wird von der Forschung nach wie vor weitgehend ignoriert, und die Frage, warum Paullus 1500 Talente kurz nach ihrer Renovierung in die Basilica investiert haben sollte, kann als ungeklärt gelten.
Eben jener L. Aemilius Paullus wird 45 v. Chr. proskribiert und flieht nach Kleinasien; die Basilica wird 34 v. Chr von seinem Sohn gleichen Namens – dann Konsul – eingeweiht.

[5] Cic. Att. 4, 17; App. civ. 2, 26; Plut. Caesar 29, 2-3.

### Die Kaiserzeit

Zwanzig Jahre später, 14 v. Chr., brennt die Basilica Aemilia vollständig nieder, das Feuer greift in einem Maße um sich, daß der nahegelegene Vestatempel und andere heilige Gebäude evakuiert werden müssen.
Der Wiederaufbau im Namen des M. Aemilius Paullus, eine Generation später, wurde von Freunden desselben und Augustus selbst finanziert[Ber65, Seite 56 ff.]. Für eine Fertigstellung der Porticus im Jahre 2 v. Chr. spricht eine Inschrift[cae][6], dem L. Caesar gewidmet, und gemeinhin der Porticus zugeordnet.

Für das Jahr 22 n. Chr. erwähnt Tacitus den Wunsch des M. Aemilius Lepidus, Sohn des M. Aemilus Paullus, an den Senat, die Basilica aus eigenen Mitteln zu renovieren und auszuschmücken[7]. Auch hier ist fraglich, ob dies eine neue Bauphase markiert.

Weitere Hinweise für Bauphasen in der Kaiserzeit fehlen; einige Implikationen archäologischer Art gibt es jedoch. Der Brand 64 n. Chr.[8] könnte das Gebäude zumindest teilweise zerstört haben, auch die flavischen Bautätigkeiten im Nordosten müßten zu einer Schließung der dort gelegenen Porticus und einer Angliederung selbiger als weiteres Seitenschiff der Basilica geführt haben. Dabei ist wohl auch ein Teil der augusteischen Außenfassade verschwunden.
Ein weiterer Brand unter Commodus zwischen 188 und 192 n. Chr. könnte wiederum das Gebäude beschädigt haben;. H. Bauer[Bau93] hat außerdem wohl Hinweise auf fünf kleinere Bauten auf den Stufen der Porticus für das 2. und 3. Jahrhundert gefunden.

### Spätantike und Nachleben

283 n. Chr., belegt durch eine große Zahl diokletianischer und maxentianischer Ziegelstempel[Bau96, Seite 32], hat die Basilica bei einem weiteren Brand wohl größeren Schaden genommen. Ein weitgehender Neubau scheint die Folge gewesen zu sein, weitere Restaurierungen unter Konstantin und Konstans sind wiederum durch Ziegelstempel belegt. Auch findet die Basilica Aemilia sich im Libellus de Regionibus urbis Romae[Nor49].

Als 410 n. Chr. Rom geplündet wurde, brannte der Bau ab. Ein Zerstörungs-

[6] Arachne-Seriennummer 132386
[7] Tac. ann. 3, 72
[8] Tac. ann. 15, 41

horizont aus Asche mit einer großen Zahl identifizierbarer Münzen[Bau77], die in die Zeit um 410 n. Chr. datieren, und die wahrscheinliche Aufgabe des Hauptschiffs sind die Folge. Im Zerstörungshorizont selber finden sich keine Bauglieder, was auf Plünderungen in der Spätantike schließen läßt.
Die Porticus allerdings wurde anscheinend wieder aufgebaut; drei Inschriften [sta] mit Nennungen der Stadtpräfekten des Jahres 416 erwähnen außerdem die Aufstellung von Statuen in einer "Basilica ilustris". Ebenfalls mit der Porticus verbunden wird eine Inschrift[lat], die Kaiser Honorius, Arkadius und den Stadtpräfekten der Jahre 418 bis 420 n. Chr. nennt. Vier Säulen aus rotem Granit werden aufgrund archäologischer Evidenz dieser spätantiken Proticus zugewiesen. Eine Ziegelmauer im Nordwesten, die Statuennischen aufweist, wurde wohl zur Verdeckung der Trümmer des Hauptschiffs errichtet.

Die letzten archäologischen Zeugnisse in dieser für ein Gebäude dieser Größe und Wichtigkeit doch sehr dünnen Quellenlage sind zwei Ziegelstempel aus einer Restaurierungsmaßnahme nach einem Erdbeben 442 n. Chr. und ein weiterer aus der ostgotischen Zeit, etwa 448 n. Chr., in der der Bau auch seine letzte literarische Erwähnung findet[Bau96, Seite 35].

### 2.2.2 Forschungsgeschichte der Basilica Aemilia

Wegen der mangelhaften Publikationslage ist eine wirklich vollständige Forschungsgeschichte der Basilica Aemilia eine Arbeit für sich; in diesem Abschnitt soll ein kurzer Überblick über die wichtigsten Grabungen und Prospektionen des Gebiets der Basilica gegeben werden, zudem ein Abriß über die Rezeption und deren Wandel im Laufe der Zeit.

#### Rezeption vor dem 19. Jahrhundert

Das Forum Romanum und damit auch die Reste der Basilica Aemilia waren ab dem Spätmittelalter respektive der frühen Neuzeit ein beliebtes Motiv für Maler und Gelehrte aus allen Ländern. Der wissenschaftliche Tatsachengehalt der Quellen ist häufig fragwürdig, da viele Darstellungen idealisiert sind, doch einige Erkenntnisse aus der archäologischen Forschung untermauern manche der Angaben zur Gestalt des Gebäudes.
Zu nennen wären da

- der Codex Escurialensis[Egg74] aus dem Jahr 1491,
- eine Vedute von Martin Hemskerk, 1532,
- die Zeichnungen und Rekonstruktionen des Architekten Pirro Ligorio im Codex Bodleianus, Oxford, circa 1550.

In der neuesten Forschung wurden vor allem die Stiche Ligorios einer Abgleichung mit der archäologischen Realität unterzogen, und einige seiner Motive scheinen recht nah an der Wirklichkeit, so die Archäologen sie aus den Bodenfunden ablesen können. Was sich endgültig daraus an Erkenntnissen ableiten läßt, bleibt allerdings noch abzuwarten.

### Grabungen

Die ersten Ausgrabungen an der Basilica begannen 1899 unter der Leitung von Giacomo Boni, deren Grabungsdokumentation allerdings nie veröffentlicht wurde.
In den 30er Jahren des 20. Jahrhunderts wurde die Basilica unter Alfonso Bartoli vollständig freigelegt und teilweise restauriert.

Die ersten (wenngleich auch spärlich) veröffentlichten Grabungsberichte stammen von einer Nachgrabung Pietro Romanellis aus den Jahren 1946-48. Dabei wurden auch Tiefgrabungen durchgeführt, die zwei weitere Säulenstellungen zutage brachten, wobei die Verbindung dieser mit den literarischen Quellen immer noch schwierig ist[Car48].
In den letzten fünfzig Jahren wurden der Fries und die der Basilica zugeordneten Barbarenstatuen häufig besprochen; die Bauornamentikforschung versuchte, die erhaltenen Bauglieder in eine Chronologie einzuordnen.

In den 70er bis 90er Jahren nahm Heinrich Bauer neue Pläne und Bauglieder auf, um zu einer Rekonstruktion des frühkaiserzeitlichen Gebäudes zu kommen. Auch versuchte er, die republikanische Baugeschichte zu beleuchten. Er publizierte seine Ergebnisse in mehreren Aufsätzen, verstarb jedoch, ohne eine abschließende Publikation zu veröffentlichen[9].

### Rekonstruktionsstand

Die bisher meistbeachtete Rekonstruktion ist die von H. Bauer[Bau88, Seite 200 ff.].
Der architektonische Gesamtkomplex der Basilica gliedert sich zum einen in die zum Forum hin ausgerichtete Porticus mit den innenliegenden Tabernen. Die eigentliche Basilica ist die Aula, die sich im Nordosten durch eine Porticus zum Marcellum hin öffnet.
Zwischen der forumsseitigen Porticus und der Aula existierten drei Durchgänge.

[9]Ein großer Teil der Pläne wurden im Rahmen des DFG-Projekts zur Basilica Aemilia digitalisiert und ist in den Arachne-Datenbanken dem Bauwerksdatensatz der Basilica Aemilia (Seriennummer 2100062) als Einzelobjekte zugeordnet.

Das an allen Seiten von Seitenschiffen begrenzte Mittelschiff wies eine Breite von drei Interkolumnien auf; in flavischer Zeit wurde die ein Stufe tiefer liegende, nordöstliche Porticus beim Bau des Templum Pacis und des Forum Transitorium in den Bau als weiteres Seitenschiff eingegliedert. Die Durchgänge von der Aula zur forumsseitigen Porticus wurden optisch durch drei auf dieser Höhe vergrößerte Interkolumnien betont.

In Bauers Rekonstruktion[10] ist die Porticus wegen der von Bauer angenommenen zwei Kapitelltypen, deren Größe um etwa 1/4 unterschiedlich ist, zweigeschossig rekonstruiert, mit einem Tonnengewölbe, über dem ein dorisches Gebälk mit Attikazone liegt.
Die zur Aula gehörigen Bauteile existieren nach Bauer in zwei Größen, so daß sich eine zweistöckige Rekonstruktion anbietet. Bauer rekonstruiert zwischen den beiden Geschossen noch ein weiteres Zwischengeschoss, welches die vielen Rankenpfeiler beinhaltet, die auf dem Gebiet der Aemilia gefunden wurden. Das Mittelschiff ist nicht von den Geschossen überspannt; es hat volle Dachhöhe.
Der Grundriß des Erdgeschosses ist ob der erhaltenen Reste relativ sicher; Bauer rekonstruiert die Säulen des Mittelschiffs mit korinthischen, die restlichen mit ionischen Kapitellen.

Die Rekonstruktion der anderen Geschosse muß als sehr unsicher gelten. Für die vorgenannte wie auch für alle weiteren Rekonstruktionen gilt: Alles, was sich nicht am Grundriß ablesen läßt, sprich die oberen Geschosse und das Treppenhaus, steht auf tönernen Füßen.

### 2.2.3 Die architektonische Situation

Im folgenden eine kurze Aufstellung der architektonischen Situation, wie sie sich heute dem Archäologen darstellt, der die Basilica Aemillia betrachtet[11]. Zunächst eine Betrachtung der in situ befindlichen Baureste, dann ein kurzer Abriß der der Basilica zugordneten ornamentierten Bauglieder, und schließlich eine kurze Aufstellung der daraus resultierenden Probleme.

[10] Die Originalzeichnungen sind in Arachne digital vorhanden und direkt dem Bauwerksdatensatz zugeordnet; Scannummern FA-S18214-18216

[11] In weiten Teilen einem Vortrag von Prof. Dr. K. St. Freyberger folgend

### Fundamente und aufgehendes Mauerwerk der Aula und der Tabernae

Die unterste Fundschicht, und damit nach der Stratigraphie die älteste, wird von Resten eines Vorgängerbaus der Basilica Aemilia eingenommen. Zu beiden Seiten der Aula kam bei den Grabungen durch G. Boni ein Steinplattenfußboden und eine von Norden nach Süden verlaufende Mauer zutage. Diese datieren früher als der erste bekannte Bau der Basilica Aemilia, der 179 v. Chr. errichtet wurde[12]. Die Räume 1 und 2 der Tabernae zeigen eine Gewandung, die aus Grotta Oscura besteht, und an den Innenseiten der Trennmauer zwischen den beiden Räumen verlaufen Abwasserkanäle. Unter dem Fußboden der Tabernae sind schräg verlaufende Mauerreste zu erkennen, die vermutlich von noch älteren Tabernen stammen. An die Rückwände der Tabernen wurde die Südmauer der Basilica in Übereinstimmung mit Liv. 40, 51.5 als eigenständige Quadermauer "post argentarias novas" errichtet[13].

Die nächste Schicht wird dem Ursprungsbau von 179 v. Chr. zugeordnet. Für die Aula gibt es dort die unmittelbar auf dem Steinplattenboden aufliegenden Fundamentquader, die wie die Rückwand der Taberne 2 aus hellgrauem Tuff bestehen. Die Oberkante der Fundamentblöcke markiert das Gehniveau der westlichen Säulenstellung des Mittelschiffs. Das Säulenjoch beträgt 6 Meter, was gegen ein steinernes Epistyl[14] spricht. Die Nordmauer ist tiefer, hat größere Quader als die Rückwand der Tabernae und besteht – wie alle anderen Mauern auch – aus grauem Tuffgestein. Die westliche Mauer wird für diese Phase im Bereich der spätantiken, schräg verlaufenden Ziegelmauer liegen. Die Ostmauer war von fünf Eingängen durchbrochen. Der Bau scheint auf allen Seiten geschlossen gewesen zu sein. Die asymmetrische Mittelschiffssäulenstellung könnte auf eine Art Vorraum im Norden hindeuten.

Über einer Brandschicht aus dem späten 2. Jahrhundert v. Chr. liegt eine neue Mauerschicht mit Quadern aus einem rötlichen Tuff. Diese finden sich in den Trennmauern der Tabernen und in der Südmauer der Basilica. Die Quader waren eisenverdübelt; 30 Zentimeter vor der Tabernenrückwand verläuft eine Frischwasserleitung, die vermutlich die von Plinius genannte Wasseruhr versorgte. In der Taberne 9[15] finden sich Reste wasserdichter Verkleidungen aus Cocciopesto, was für eine Lokalisierung an dieser Stelle sprechen würde.

[12] Die Pläne: Arachne-Seriennummer BT2; speziell Scan-Nummer FA-S18011
[13] Arachne-Seriennummern BT6 und 2100062, dort Scan FA-S18211
[14] griechisch: "das auf den Säulen liegende"
[15] Arachne-Seriennummer BT6, Scan FA-S18130

Für die nächste Bauphase belegen die oben genannten Münzbilder einen zweigeschossigen Ausbau des Innenraumes mit den besagten Familienbildern. Für diesen Neubau mit steinernem Epistyl wurde eine Verkürzung der Säulenjoche notwendig, was eine größere Anzahl Säulen und eine Verschiebung des westlichen Seitenschiffs nach Osten nötig machte. Damit erreichte die Basilica ihre endgültige bauliche Gestalt, die vom augusteischen Bau dann in Marmor ausgeführt wurde[16]. Die Fundamente dieser neuen Säulen wurden aus Travertin gefertigt. Auch die Nordmauer erhielt in dieser Phase über den Fundamentblöcken eine Reihe hoher Travertinblöcke und eine darüber angesetzte Wand aus rötlichem Aniene-Tuff. Diese Wand war von Türen durchbrochen. Die Portiken und damit die Tabernen wurden mit einem weiteren Geschoß auf etwa neun Meter erhöht. Auch die Trennwände der Tabernen wurden mit neuen Fundamenten versehen und von 60 auf etwa 90 Zentimeter verbreitert. An die Stirnwände der Portiken wurden monumentale Travertinblöcke angesetzt, die den Fundamentblöcken der Säulen ähnelten.

Für den augusteischen Neubau wurde, wie erwähnt, der Grundriß beibehalten und große Teile der Substanz mit Marmor verkleidet. Bauer schlägt in seiner Rekonstruktion[17] ein tonnenüberwölbtes Attikageschoß vor; nach neuen Erkenntnissen sind jedoch die Aufleger auf den Architraven zu schmal, um eine solche Last zu tragen, und auch die Zugehörigkeit der Blöcke, die er dort hin rekonstruiert, ist mehr als umstritten. Die Säulen wurden durch solche aus Africano ersetzt, und Funde marmorener Dachplatten würden ebenfalls gut als Dachdeckung zu einem augusteischen Bau passen. Auch bei der Rekonstruktion der Tabernae ist die von Bauer vorgeschlagene Zweistöckigkeit eher unwahrscheinlich. Zum einen würde dies der Aula jegliche Belichtung von Süden nehmen, zum anderen wiesen neue Forschungen von Freyberger und Lipps nach, daß es sich bei den “kleineren” Kapitellen, aus denen Bauer eine zweigeschossige Rekonstruktion erstellt, um beschädigte Exemplare der einen, bekannten Ordnung handelt. Einer der Pfeiler der Porticus steht bis heute in situ an der Südseite. Nicht nur die Porticus und die Aula wurden in Marmor neu aufgebaut, auch die spätrepublikanischen Tabernae wurden unter Augustus mit Marmorplatten verkleidet.

### Bauornamentik

Dieser Punkt bezieht sich maßgeblich auf den augusteischen Bau, da von der Bauornamentik der Vorgängerbauten nichts von Substanz vorliegt. Viele der Bauglieder, die durch solche aus Marmor ersetzt wurden, wurden wohl – wie

[16] Arachne-Seriennummer 2100063, Scan FA-S18211
[17] siehe dort

so oft in allen Zeiten – als Spolien in anderen Gebäuden zweitverwertet. So finden sich Spolien der kaiserzeitlichen Basilica an der Villa Torlonia.

In der Aula finden wir eine zweigeschossige Säulenstellung, bei der das Untergeschoß nach momentanem Forschungsstand eine ionische, das Obergeschoß eine korinthische Ordnung hatte. Für beide Ordnungen wurden die zuvor aus Tuff bestehenden Säulen durch teurere, prachtvollere aus Africano ersetzt. Die Travertinbasen sind wohl noch die des Vorgängerbaus. Das Gebälk der unteren Ordnung besteht aus Architrav, Friesband und einem Konsolengesims. Hier offenbart sich direkt die erste Schwierigkeit: der Fries mit der Darstellung der Gründungsgeschichte Roms wurde bisher meist in diese Ordnung rekonstruiert, doch eine Neuaufnahme der Bauteile widerspricht dieser Annahme. Dübellöcher erlauben eine Rekonstruktion der Architravbreite, und diese widerspricht der Höhe des Frieses. So muß für den Fries noch eine neue Position gefunden werden.
Die Säulen der zweiten Ordnung trugen korinthische Kapitelle; darüber lag ein Gebälk mit einem Faszienarchitrav, Lotus-Palmette-Fries und Konsolengesims.
Sowohl die ionischen wie auch die korinthischen Kapitelle lassen sich laut Lipps[Lip05, Seite 20 ff.] in zwei Gruppen einteilen, wobei die eine Gruppe zum Mittelschiff zu gehören scheint und die andere Gruppe jeweils dem Seitenschiff zuzuordnen wäre.

Desweiteren erhalten, aber noch nicht wissenschaftlich bearbeitet, sind eine große Anzahl Architravfragmente[18], einige Statuenreste und Pilasterkapitelle. Die qualitativ äußerst hochwertigen Rankenpfeiler[19], die auch der Basilica zugeordnet werden, sind hingegen gut publiziert[MF99, Seite 67 ff.]. Diese zu verorten ist eine der aktuellen Aufgaben der Archäologie, wobei hier erstmals mit einer Datenbank und elektronisch-architektonischen Modellen gearbeitet wird. Die Aufbereitung der Daten für das hier vorliegende Projekt wird sicher geschehen, aber zum Zeitpunkt der Abgabe dieser Arbeit noch bei weitem nicht abgeschlossen sein.

### Rekonstruktionsprobleme

Die Probleme bei der Rekonstruktion der Basilica Aemilia sind vielfältig: Zum einen wäre da die ungenügende Publikationslage, da kaum einer der Forscher, die sich teilweise über Jahre hinweg mit der Basilica Aemilia beschäftigten, eine Publikation seiner Arbeit vorlegte. Damit zusammenhängend ergibt

[18] Zum Beispiel Arachne-Seriennummern 132265, 132319, 132845...
[19] Arachne-Seriennummern 68272, 132903-8

sich ein weiteres Problem, mit dem sich die aktuelle Forschung an der Aemilia auseinanderzusetzen hat: Viele Bauglieder wurden immer wieder bewegt, es gab einige nicht dokumentierte Rekonstruktionsversuche am Gebäude, und die wenigen wirklich publizierten Artikel sind oft nicht überprüfbar.
Auch die mittlerweile in extensio erforschte literarische Quellenlage gibt besonders für den kaiserzeitlichen Bau kaum Hinweise, obwohl dieser mehrere Jahrhunderte das Stadtbild am Forum Romanum geprägt hat.

Hier nun der Bogen zum eigentlichen Gegenstand dieser Arbeit: Mit einem Werkzeug, welches es dem Wissenschaftler ermöglicht, seine Ergebnisse nachvollziehbar zu speichern, und es dem nächsten Wissenschaftler ermöglicht, dieses nachzuvollziehen und mit seinem Wissen zu erweitern, wäre der Archäologie ein wichtiges Mittel zur Dokumentation und Rezeption besonders schwieriger Stücke gegeben, und dem interessierten Laien eine Möglichkeit, sich besser zu verdeutlichen, was der Wissenschaftler warum wie gemacht oder gemeint hat.

# Kapitel 3

# Informationstechnische Grundlagen

Dieses Kapitel beschäftigt sich mit den theoretischen Grundlagen und der Betrachtung bereits bestehender Systeme, die auch das Ziel haben, ein 3-D-Modell mit einer Datenbank zu verbinden.
Die theoretischen Grundlagen beinhalten eine Analyse der Anforderungen, die Archäologen als Zielgruppe für dieses Projekt haben, und einen Exkurs in die Benutzbarkeit neuer Systemen unter dem Gesichtspunkt der Kognition. Zudem sollen Grundlagen wie Formatfragen und eine Bestandsaufnahme der bereits bestehenden und im Rahmen des Projekts verwendeten Ressourcen geklärt werden.

## 3.1 Semantik und Methode

Die klassische Archäologie ist eine der Geisteswissenschaften, bei der - wie bei fast allen Wissenschaften - die Semantik, also die Benennung von Objekten und die Hierarchie der Bezeichnungen von großer Wichtigkeit ist. Eins der Probleme mit diesen archäologischen Begrifflichkeiten ist der Mangel an Eindeutigkeit - es gibt keinen festgelegten Bezeichnungskanon, keinen verbindlichen Thesaurus. Trotzdem weiß jeder Archäologe sofort, wenn eine Benennung falsch ist, wenn Begriffe außerhalb der ihnen zugedachten Domäne mit anderen Bedeutungen belegt ist.
Deswegen ist es für ein Projekt, welches sich mit computergestützter Dokumentation von Baukomplexen beschäftigt, sehr wichtig, diese Begriffe entweder genau richtig anzuwenden, oder vollständig auf sie zu verzichten und die Bedeutung von Zeichen auf einer nichttextuellen Ebene zu klären.

Diese Arbeit versucht, dem zweiten Ansatz zu folgen, was sich bei der Visualisierung und der Anbindung an eine bestehende Datenbank, bei der die textuelle Semantik ständig überprüft wird, der Weg der Wahl ist. So wird durch die rein visuelle Abstraktion jeder Benutzer dazu angehalten, sich kritisch mit dem Gegenstand an sich und nicht mit der Richtigkeit von Bezeichnungen auseinanderzusetzen.

### 3.1.1 Archäologische Erfordernisse

Eine Grundanforderung an das Projekt ist eine intuitive Bedienbarkeit, die vom archäologischen Benutzer keinerlei besondere Computerkenntnisse verlangen sollte. Intuitiv muß sich demjenigen, der mit der Datenbank schon gearbeitet hat, erschließen, wie die Erweiterung um eine Visualisierung der Suchergebnisse und das Suchen über das 3-D-Interface funktioniert.
Diese sollte im besten Falle so aussehen, daß die Teile im 3-D-Modell, die anklickbar sind, sich optisch vom Rest des Modells abheben. Desweiteren sollte es ein visuelles Feedback bei Auffindung eines zum Bauwerk gehörigen Suchergebnisses geben, welches über das reine Hervorheben der Bauteilgruppe hinausgeht - es ist nicht nötig, daß der Benutzer im Browser informiert wird, daß sich etwas im Modell geändert hat, aber es wäre wünschenswert, wenn sich mit dem nächsten Blick auf das 3-D-Fenster zeigte, daß etwas passiert ist.

Eine weitere, wichtige Grundanforderung ist die Portabilität des Systems. Es sollte ohne größere Eingriffe in den Code oder den Datenbestand (sowohl was die Objektdaten in der Datenbank als auch was das 3-D-Modell angeht) auf möglichst allen gängigen Plattformen laufen, namentlich MacOS X, Linux und Windows.
Auch essentiell ist die Erweiterbarkeit um neue Projekte, die dieselbe Engine benutzen sollen. Es muß einfach sein, beispielsweise ein neu erstelltes Modell der Ara Pacis einzupflegen und die Daten in der Datenbank so aufzuarbeiten, daß sie sich in der dreidimensionalen Bedienoberfläche darstellen lassen.
Zudem ist die Möglichkeit, unterschiedliche Forschungsmeinungen ausdrücken zu können, von großer Bedeutung für eine mögliche Produktionsfassung dieses Projekts, denn eine archäologische Datenbank, und damit auch alle mit ihr verbundenen Module, sollte immer möglichst viele oder, wo das möglich ist, alle Meinungen zu strittigen Themen darstellen, damit sich der Benutzer ein eigenes Bild machen kann. Dies soll hier als mögliche Weiterentwicklung des Projekts besprochen werden, da die Implementierung im Rahmen dieser Arbeit zu aufwendig gewesen wäre.
Portabilität und Erweiterbarkeit werden im Kapitel zur Realisierung weiter

ausgeführt, die intuitive Bedienbarkeit und die Erfordernisse der Quellenkritik folgen in diesem Kapitel.

### 3.1.2 Bedienoberfläche und Kognition

So, wie es eine kognitiv bedeutend geringere Anstrengung ist, einen Stift und ein Blatt Papier zu benutzen, als einen Computer, so sollte auch ein jedes Interface eine geringe kognitive Schwelle aufweisen[AC06, Seite 27]. Deswegen ist die Verbindung mit einer bestehenden Datenbank in vielerlei Hinsicht sinnvoll: So wird der Entwicklungsaufwand deutlich geringer, da die Datenbank in ihrer Struktur bereits existiert. Auch die Benutzung des bestehenden Interfaces hat in kognitiver und bedienergonomischer Weise einen großen Vorteil: Wissenschaftler oder Laien, die bereits mit der Datenbank gearbeitet haben, müssen sich nicht an eine neue Benutzeroberfläche gewöhnen, sondern können alle Abfragen und Funktionen des Datenbankteils so benutzen, wie sie es gewohnt sind.

Das Bedienen eines 3-D-Interfaces mit seinen sechs Freiheitsgraden stellt andere Anforderungen an den Benutzer als ein Datenbank-Interface. Es gibt viele Ansätze[1], und die meisten dieser Bedienschemata haben für genau die Anwendung, aus der sie stammen, auch ihre Bewandtnis.
Zu unterscheiden ist auch zwischen den unterschiedlichen Zwecken der 3-D-Steuerung. Es gibt drei wichtige Kategorien: Navigation, Auswahl und Systemsteuerung. Viele Erwägungen verschiedener Steueransätze hängen mit den unterschiedlichen Ein- und Ausgabegeräten zusammen, die für dreidimensionale Umgebungen zur Verfügung stehen[2], doch da sich diese Applikation auf die Verwendung von klassischen Büro-PCs beschränkt, ist nur ein kleiner Teil dieser Geräte überhaupt verfügbar. Von der gängigsten Konfiguration ausgehend, dem zweidimensionalen Monitor mit Maus und Tastatur, gibt es betreffs der Navigation mehrere Konzepte, die sich anbieten[DB01, Seite 98 ff.]:

- die aus CAD-Programmen und Modellern bekannte Perspektivenmanipulation, “Manual Viewpoint Manipulation”,
- die klassische First-Person-Shooter(FPS)-Steuerung, vgl. Doom usw., auch “Steering” genannt,
- kontextsensitive, zielgerichtete Bewegung, “Target-based Travel”.

[1] Eigentlich fast so viele, wie es dreidimensionale Interfaces gibt
[2] 3-D-Monitore, CAVEs, Handschuhe, Zeigegeräte und so weiter

Einige Annahmen vorweg: Sowohl die Maus als auch die Tastatur sind in der Lage, einerseits diskrete Signale, wie das einmalige Drücken einer Taste, zum Beispiel für eine Auswahl, andererseits aber auch kontinuierliche Signalströme wie das Bewegen des Mauszeigers zu erzeugen.
Dies führt auch dazu, daß Eingabegeräte mit einer kleinen Anzahl von Freiheitsgraden mit Hilfe einer diskreten Tastenkombination mehr Freiheitsgrade emulieren können. So könnte man zum Beispiel über die Mausbewegung die Blickrichtung steuern, durch Drücken der mittleren Maustaste allerdings die $y$-Achse zum Zoomen verwenden. Grundlegend läßt sich jede Steuerung in zwei Komponenten teilen: die motorische Reisekomponente, um von einer Perspektive zu einer anderen zu gelangen, und die kognitive der Wegfindung. Beide sind in unterschiedlichem Maße für die drei Kategorien der Navigation relevant.

"Erkundung" ist das Erfahren einer Umgebung, wobei die Erfahrung der Selbstzweck ist - in diesem Projekt das Steuern der Person durch das Gebäude, mit einem Viewport, der auf Höhe der maßstabsgetreuen Augen des Benutzers liegt. So kann der Benutzer den Raumeindruck des Gebäudes auf sich wirken lassen. Hier eine Nebenbemerkung: Damit ein Raumeindruck wirklich entstehen kann, muß es außer der Augenhöhe noch einige, aus der realen Welt bekannte Fixpunkte für die kognitive Größenfeststellung geben – das kann ein Stuhl sein, der mitten im Raum steht, oder ein Personenmodell, etwas, was jeder kennt und womit jeder einen festen Maßstab verbindet.
"Suche" ist das gezielte Hinsteuern auf ein Objekt. Wichtige Kriterien sind auf der einen Seite, nicht die Orientierung zu verlieren, auf der anderen Seite die Kenntnis des Ziels. Komplexere Modelle würden für diese Anforderungen Navigationshilfen benötigen, gegebenenfalls in der Art, wie First-Person-Shooter[3] in einer der oberen Ecken häufig eine Karte der Umgebung eingeblendet haben. Hier käme auch das Element der Systemsteuerung im dreidimensionalen Umfeld zum Tragen: das Umschalten zwischen Kartenansicht, in der man auch Navigieren können muß, und die reguläre Reise durch die Umgebung. Auch ist es wichtig, das Auswahlwerkzeug aus der Suche heraus schnell verfügbar zu haben.
Die letzte Kategorie ist das Manövrieren. Dies beschreibt die kleinräumige Veränderung der Perspektive, um beispielsweise ein Objekt exakt auswählen zu können oder einen idealen Blickwinkel für ein Standfoto, welches einen gewissen Aspekt der Rekonstruktion besonders gut hervorhebt, zu haben. Diese Form der Navigation wird jedoch in dieser Arbeit nicht angewendet werden, da für den Zweck selbiger die Erkundung eine hinreichend exakte

[3]Doom, Quake und so weiter

Steuerung darstellt.

Bezüglich des eigentlichen Projekts bedeutet dies, daß Steuerungs- und Navigationskonzepte für die unterschiedlichen Anforderungen zusammengebracht werden müssen. Die erste wichtige Funktion des Programms ist die Erkundung des eigentlichen Gebäudes der Basilica Aemilia. Für diese Tätigkeit bietet sich das "Steering" mit Keyboard und Maus an. Die Tastenbelegung ist an der gängigen FPS-Belegung angelehnt; "w" steuert vorwärts, "s" rückwärts, und "a" und "d" links und rechts. Bei gehaltener Umschalttaste wird die Geschwindigkeit erhöht, so daß es dem Benutzer scheint, er würde rennen. Die Maus steuert den "Kopf", und projiziert auf die zweidimensionale Lauffläche bewegt man sich immer dahin, wohin man schaut. Da das Modell - und auch die 3-D-Engine - keine Treppen besitzt, kommt man mit Hilfe der Tasten 1-3 an vordefinierte Punkte in den unterschiedlichen Geschossen. Dort überschneiden sich die Navigationsarten; aus dem reinen "Steering" wird eine zielgerichtete Bewegung, die auch für das wissenschaftliche Verständnis der Rekonstruktion wichtig ist. Desweiteren ist es wünschenswert, daß man ideale Ansichtspunkte für die unterschiedlichen Bauteilgruppen erreichen kann. Dies sollte aber nicht in Form einer "Teleportation" erfolgen, da dies den Nutzer in puncto eigener Position im Raum leicht verwirren kann, sondern mit einer Art Hinfliegen, welches Bowman et al. auch als "Magic Carpet Ride" bezeichnen.

### 3.1.3 Benutzbarkeit und Quellenkritik

Wenn nun sowohl die Installation problemlos erfolgen kann, es keinen Unterschied macht, welche Plattform ein Benutzer hat, und die Art des Anklickens und der Darstellung von Suchergebnismengen plausibel und verstanden ist, kommt der Punkt, an dem die wissenschaftlich-archäologische Komponente einsetzt.

Quellenkritik ist in jeder Wissenschaft ein zentraler Punkt, so auch in der Archäologie, besonders bei strittigen Themen, wie etwa der Rekonstruktion eines Gebäudes, von dessen Gestalt nur noch die Grundmauern bekannt sind. Es fängt bereits bei der gefundenen Bauornamentik an: Was davon wird von wem warum diesem Gebäude zugeordnet? Welche Schlüsse betreffs der Höhe oder der Gewölbestruktur werden daraus gezogen?
Diese Fragen muß ein System, welches Rekonstruktionen visualisieren und im Falle fast nicht mehr vorhandener Gebäudestruktur auch bewerten muß, in Betracht ziehen. Es muß für quellenkritische Arbeit möglich sein, verschiedene Rekonstruktionen, und damit auch verschiedene Modelle, mit den

dieser speziellen Rekonstruktion zugrunde liegenden Bauteilverortung nebeneinander zu betrachten. Um dies zu erreichen, müßte der vorhandene Design- und Funktionsprototyp um mehrere Modelle, die die unterschiedlichen Rekonstruktionsvorschläge reflektieren, bereichert werden, und es sollte möglich sein, sich diese Modelle im selben Fenster nebeneinander anschauen und die Verortung der vorhandenen Bauteile, auf denen die Rekonstruktion beruht, visuell analysieren zu können.

Hier kommt nun wieder die Art der Navigation ins Spiel. Nachdem der Benutzer sich einen Eindruck des Gebäudes verschafft hat, indem er "hindurchgegangen" ist, wird es wichtig, daß er sich die für die Rekonstruktion relevanten Bauteile aus verschiedenen Perspektiven ansehen kann. Dies verlangt nach einer zielgerichteten Perspektivenveränderung, die dem Nutzer die Bauteile aus einer Perspektive zeigt, die sowohl eine möglichst vollständige Ansicht der Bauteilgruppe als auch der anderen, für die Plazierung an genau dieser Stelle wichtigen Bauteile zeigt. Die einfachste Methode wäre eine Art "Teleportation", sprich, der augenblickliche Ansichtswechsel von der zuvor eingenommenen Position hin zu der idealen, die im Programmcode oder in der Datenbank hinterlegt ist. Solch ein augenblicklicher Positionswechsel hat aber seine Nachteile. Die Orientierung, das Bewußtsein, wo genau man sich im Gebäude befindet, leidet, wenn man keine Rückmeldung in Form einer kontinuierlichen Bewegung hat. Ein Perspektivenwechsel sollte dem Nutzer immer die Möglichkeit geben, nachvollziehen zu können, wie er an den neuen Aussichtspunkt gekommen ist. Grundlegend gibt es dafür zwei Möglichkeiten, da der Ausgangspunkt, von dem die Bewegung her erfolgt, nicht vorhersagbar ist. Die erste Möglichkeit, die ob ihrer Komplexität für dieses Projekt nicht gewählt wurde, wäre eine kollisionsbewußte Wegfindung, sprich, eine Bewegung zwischen den Gebäudeteilen hindurch, ohne eines zu berühren. Die zweite, weitaus einfachere Möglichkeit, die dem Benutzer trotzdem ein Gefühl der Positionsänderung gibt, ist die, ohne Rücksicht auf die Baumasse den Benutzer zu dem neuen Aussichtspunkt zu bewegen. Zweckmäßigerweise sollte auch hier auf abrupte Geschwindigkeits- und Sichtrichtungswechsel verzeichtet werden, so daß die gesamte Bewegung erst beschleunigen, dann abbremsen muß. Das genauere Konzept wird im Realisierungsteil der Arbeit angesprochen.

## 3.2 Ähnliche Systeme

### 3.2.1 CAD und Datenbanken

Die Idee, Datenbanken mit Modellen zu verbinden, ist nicht neu. Eine gebräuchliche Technologie sind zum Beispiel Geografische Informationssysteme, kurz GIS[HS97]. In diesen werden Karten mit Daten verbunden, so daß es möglich wird, beispielsweise statistische Auswertungen über Streuung, Sichtbarkeitsanalyse und physikalisch-modellgestützte Prognosen zu berechnen. Diese GIS arbeiten jedoch immer mit sogenannten 2 1/2-D-Modellen. Das heißt, daß beispielsweise Oberflächendaten zwar mit Höhenwerten versehen und auch als dreidimensionale Landschaft darstellbar sind, Modelle mit einem Innenraum - wie zum Beispiel Gebäude - jedoch nicht betreten oder innenliegende Teile derer auf einen Datensatz referenziert werden können. Außerdem liegt ein großer Schwerpunkt bei geographischen Informationssystemen auf der Möglichkeit, statistische Berechnungen durchführen und grafisch darstellen zu können, was mit dem hier beschriebenen Ansatz eines Bauwerksinformationssystems nicht viel gemein hat.

### 3.2.2 Rechnergestütztes Anlagenmanagement

Unter Rechnergestütztem Anlagenmanagement[4] versteht man in der Wirtschaft die Verwaltung von Gebäuden und den in ihnen befindlichen Gegenständen mit Hilfe einer Datenbank und eines Modells des Gebäudes. Die Auswahl an Produkten ist recht groß[caf06]; die Preise für allein die nicht auf die speziellen Bedürfnisse eines Kunden zugeschnittenen Grundversionen sind hoch. Auch sind solche Systeme, wie sie zum Beispiel für die Raumverwaltung oder die von Schlüsseln benutzt werden, nicht ohne Weiteres für archäologische Bedürfnisse geeignet.
So konnte bei dem einzigen zur Evaluation zu Verfügung stehenden Produkt ArchiFM[arc06a] keine externe Datenbank angesprochen werden, sondern nur die mit dem System proprietär mitgelieferte. Außerdem war es nicht möglich, Bauteile mit Datensätzen zu verknüpfen, sondern nur Inventargegenstände oder abstrakte Entitäten wie zum Beispiel Schlüssel, womit dieses Produkt keiner der Anforderungen an das Projekt standhielt.

[4]Gebräuchlicher ist - auch im Deutschen - allerdings der Begriff Computer Aided Facility Management (CAFM), der sich auch als Abkürzung direkt in die Familie der 3-D-Programme (zum Beispiel CAD, CAM) einordnet.

### 3.2.3 Projekte in der Kunstgeschichte und Archäologie

Abgesehen von geographischen Informationssystemen, die in der Archäologie, und dort besonders in der prähistorischen, recht verbreitet sind, gibt es bisher nur wenige Projekte, die eine Datenbank mit einem 3-D-Modell verbinden wollen. Es gibt wohl pädagogische, häufig in Museen eingesetzte, individuell erstellte Programme, die fest eingegebene Daten mit einem Rundgang durch eine virtuelle Rekonstruktion verbinden, aber für den Anschluß einer sich ständig aktualisierenden Datenbank an ein 3-D-Modell gibt es noch kein funktionsfähiges System.

Zu nennen ist der Ansatz der TU Cottbus[Hen], die in Zusammenarbeit mit dem Deutschen Archäologischen Institut eine Raumbuchdatenbank für die Ausgrabungsprojekte auf dem Palatin in Rom[Deu06b] und die Stadtgrabung in Baalbek[Deu06a] erstellen. Auch hier sollen dreidimensionale Rekonstruktionen und Bauaufnahmen mit Datensätzen aus einer Datenbank verknüpft werden. Dieses Projekt basiert auf VRML, ist aber zur Zeit[5] noch nicht aus dem Prototypenstadium fortgeschritten.

## 3.3 Formate und Ansätze

Wie bei jedem Projekt, besonders bei Aufgaben, die mehrere Technik- und Wissensdomänen miteinander zu verbinden suchen, versucht auch diese Arbeit, sich durch die Benutzung von Standards kompatibel mit zukünftiger Technologie zu halten und den Arbeitsaufwand überschaubar zu halten.
Die unterschiedlichen Standards, die in Betracht gezogen werden müssen, ergeben sich aus den Projektanforderungen: dreidimensionale Modelle, Netzwerkkommunikation und Datenbanktechnik.

### 3.3.1 3-D

Die Welt der 3-D-Formate ist groß und vielfältig. Fast jeder Hersteller hat sein eigenes, proprietäres Binärformat, auch die Menge der Austauschformate ist sehr groß. Einige Datenformate – wie zum Beispiel das AutoDesk-Format DXF[Aut06] – haben sich relativ weit verbreitet und können von vielen gängigen Programmen geschrieben und gelesen werden. Andere Formate, wie das WaveFront OBJ[obj06] haben den großen Vorteil, auch von Menschen lesbar zu sein und eine sehr übersichtliche Struktur zu besitzen, was

[5] August 2006

das Programmieren einer Laderoutine bedeutend vereinfacht. Viele Binärformate können zwar bedeutend mehr Informationen speichern, sind aber entweder nur schlecht oder gar nicht dokumentiert. Ein Beispiel für ein volldokumentiertes, aber unhandliches Binärformat stellt die Dateiart .blend dar, die vom freien 3-D-Modeller Blender als eigenes Format eingesetzt wird: In der Binärdatei werden nicht nur übliche Sachen wie Vektoren, Polygone, Texturen, Lichter und Koordinaten gespeichert, sondern auch alle Einstellungen des Programms zum Zeitpunkt des letzten Sicherns. Bei früheren Versionen (prä-2.0) speicherte sich das ganze Programm mit, so daß man die Datei auf jedem Rechner desselben Betriebssystems wie das des erstellenden Rechners ohne weitere Programminstallation öffnen konnte.

### 3.3.2 Netzwerktechnologien

Das gesamte Projekt setzt auf der internetbasierten Datenbank Arachne auf, so daß dieser Abschnitt recht kurz zu fassen ist. Die Anfragen an die Datenbank werden als PHP-CGIs abgesetzt, die Datenbank ihrerseits schickt valides HTML an den Browser respektive den Proxy zurück. All das sind wohlbeschriebene und seit Jahren verwendete Standards. Die Probleme, die auftauchten, waren, wie es später beschrieben wird, anderer Natur.

### 3.3.3 Datenbanken

Eines der Ziele dieser Arbeit ist die Möglichkeit, das Projekt auch auf andere als die hier exemplarisch verwendete Datenbank aufzusetzen. Deswegen ist es wichtig, daß es eine Kommunikationsform gibt, die von quasi jeder webbasierten Datenbank ausgeführt werden kann, ohne große Änderungen im Code vornehmen zu müssen. Eine ausführlichere Darstellung der wenigen für dieses Projekt an der Datenbank notwendigen Änderungen findet sich im Realisations-Teil.

## 3.4 Bestehende Grundlagen

Da in der Informationstechnik, und im Rahmen einer Magisterarbeit im besonderen, das Rad weder immer neu erfunden werden soll noch kann, setzt dieses Projekt auf einer Vielzahl von bestehenden Komponenten auf.
Dies zeigt auch, wie wiederverwendbar die Komponente an sich ist – die Wahl von Arachne als Datenbank war sicher nicht nur aus technischer Sicht erfolgt, doch steht Arachne als klassische MySQL/Apache/PHP-Lösung stellvertretend für eine Vielzahl anderer Datenbanken, auf die dieses Projekt auch –

mit nur geringen Änderungen im Code – aufsetzen kann.

### 3.4.1 Die Bilddatenbank Arachne

Arachne[For06] ist die Datenbank und die kulturellen Archive des Forschungsarchivs für Antike Plastik des Archäologischen Instituts der Universität zu Köln und des Deutschen Archäologischen Instituts (DAI). Anfangs wurde Arachne als FileMaker-Lösung entwickelt; seit Anfang 2005 jedoch läuft die komplette Datenbank auf einer MySQL/Apache/PHP-Grundlage.

Bei der Umstrukturierung von Arachne hin zu einer Open-Source-Lösung wurden viele Paradigmen der objektorientierten Programmierung umgesetzt und auf das Konzept einer relationalen Datenbank abgebildet. So sind sämtliche SQL-Befehle in einer Abstraktionsschicht, dem so genannten Data Access Object[Ger06] (DAO) hinterlegt, so daß alle Operationen auf die Tabellen als Objektmethoden der einzelnen Datenobjekte ablaufen und der Programmierer nicht mit der Datenbank direkt interagieren muß. Dies hat unter anderem den Vorteil, daß für die Umsetzung der von diesem Projekt benötigten Erweiterung, die sich auf archäologische Objekte bezieht, nur einige Zeilen Code geändert werden mußten, damit bei den betroffenen Datensätzen das entsprechende HTML-Tag in den an den Browser übermittelten HTML-Code eingebettet wird.

### 3.4.2 Verwendete Bibliotheken

Kein Programmierer erfindet für jedes Projekt das 3-D-Interface oder die Funktionen der Netzwerkkommunikation neu, und so gibt es eine große Zahl frei verfügbarer Bibliotheken, die häufig benötigte Funktionen bündeln und in einfacher Weise zur Verfügung stellen.

#### Asmoday

Asmoday[Gra06] ist ein Werkzeug für die komfortable und schnelle Erstellung von 3-D-Anwendungen in FreePascal. Ursprünglich nur für Linux verfügbar, läuft die Bibliothek nun auch unter OS X (PowerPC- wie auch Intel-basiert) und – mit einigen Einschränkungen – auch unter Windows. Es bietet eine rudimentäre, leicht zu erweiternde 3-D-Engine, einen dreidimensionalen Editor[6] zur Lichtsetzung und Objektpositionierung und läßt sich als Komponente in die Lazarus IDE einbinden. Die Engine bietet zum Beispiel verschiedene Kameraarten, geometrische Primitiven und der Editor eine Laderoutine für

[6] Bisher nur unter Linux und unter OS X auf X11

WaveFront-.obj-Dateien.

Asmoday befindet sich sicher noch im Alpha-Stadium der Entwicklung, doch reichte es für die Erstellung einer plattformübergreifenden, einfachen OpenGL-Engine wie dieser vollkommen aus.

### Synapse

Synapse[Geb06] fing ursprünglich als eine nur für Windows verfügbare Netzwerkbibliothek an, die auf Winsock aufsetzte und eine Lücke zu schließen suchte hinter den weit verbreiteten Lösungen, die asynchrone Methoden benutzen, und eine Bibliothek aufbaute, synchronen Netzwerkverkehr über das Blockieren von Sockets zu ermöglichen. Die ersten Versionen waren für Delphi auf Windows und später für Kylix unter Linux gedacht, doch mit dem Reifen von Lazarus zu einer ernstzunehmenden Alternative wurde die Bibliothek auch dorthin portiert und läuft nun, nach einigen Anpassungen, auch unter OS X. Sie stellt recht einfach zu benutzende Funktionen für synchronen TCP/IP-Verkehr, besonders zur Server- und Client-Erstellung für das HTTP-Protokoll zur Verfügung. Ein großer Teil der weiteren TCP/IP und UDP-Funktionen sind auch vorhanden, werden im Rahmen dieses Projekts aber nicht benutzt.

# Kapitel 4

# Realisierung

Dieses Kapitel befaßt sich mit der Umsetzung des Vorgenannten in einen Design- und Funktionsprototypen und mit den benutzten Werkzeugen, deren Stärken und Schwächen und dem benötigten Anpassungsaufwand.

## 4.1 Umgebung

Unter einer Programmierumgebung wird gemeinhin die Kombination aus Compiler, Editor, Versionierungs- und Projektmanagementsoftware sowie den Programmbibliotheken verstanden. Im besten Fall sind diese Komponenten in einem sogenannten Integrated Developer Environment (IDE), einer integrierten Entwicklungsumgebung, zusammengefaßt. Dieser Abschnitt behandelt die Evaluation unterschiedlicher Möglichkeiten und die Gründe für die schlußendlich gewählten Komponenten.

#### Exkurs: MacOS X als Entwicklerplattform

Da das für diese Arbeit erstellte Programm fast ausschließlich auf zwei Rechnern mit MacOS X 10.4[1] entstand und Programme und Bibliotheken benutzte, die nicht ohne weiteres für selbiges verfügbar waren, hier nun einige Worte zur grundlegenden Struktur des Apple-Betriebssystems und den behobenen Problemen mit nicht für dieses System entwickelten OpenSource-Lösungen.

Nach der großen Umstellung von MacOS 9 auf OS X begann für Apple die Zeit, in der die Firma ein zeitgemäßes Betriebssystem vorweisen konnte. Als unterste Ebene – ganz wie in Tanenbaums Lehrbuch[TW97] – fungierte

[1] Apple PowerBook G4, 1,33 GHz; nach dessen Funktionsunfähigkeit ein MacBook mit einem 2 GHz Intel CoreDuo-Prozessor

ein Mach Microkernel, auf den ein FreeBSD 4.4 "Userland"[2] aufgesetzt ist. Zusätzlich gibt es eine größere Auswahl der gängigsten GNU-Tools wie emacs und den gcc – soweit die OpenSource-Komponenten. Auf dieses FreeBSD setzen nun grundlegende Systembestandteile wie CoreGraphics, OpenGL oder QuickTime auf, darauf dann zwei Bibliotheks-Umgebungen, die – zum Beispiel mit ihrer Thread-Verwaltung, siehe dort – teils bis tief in das BSD hereinreichen. Die historisch ältere ist Carbon, die noch auf großen Teilen des alten OS 9 aufbaut, die neuere ist Cocoa, in der sich viele der alten NextStep-Komponenten wiederfinden[3]. Nicht von ungefähr wird Carbon eher (aber nicht ausschließlich) für prozedurale Programmierprojekte benutzt, während Cocoa ein rein objektorientiertes Framework ist.

Für die OS X-Version des Programms wurde OpenGL als Darstellungskomponente gewählt. Mit dieser gab es nur geringfügige Probleme, da Apple die gesamte grafische Oberfläche als ein OpenGL-Objekt behandelt und das Fenster, in dem das Programm läuft, deswegen nicht "rootless" sein kann – dies wurde durch eine kleine Veränderung am Asmoday-Code durch seinen Entwickler behoben, die es dem OpenGL-Fenster erlaubte, als Tochterobjekt der gesamten Oberfläche zu laufen.

### 4.1.1 Sprache

Eines der Hauptziele eines solchen Projekts, bei dem es um die Strukturierung und Zugänglichmachung von wissenschaftlichen Daten geht, sollte sein, die Anforderungen für die Benutzung des Programms möglichst gering zu halten. So sollte das Programm nicht nur auf auch etwas älterer Hardware laufen, sondern auch möglichst unabhängig vom benutzten Betriebssystem funktionieren.

So ist dann eine Voraussetzung, um das Programm nicht für jedes Betriebssystem neu schreiben zu müssen, daß es entweder auf jedem gängigen Betriebssystem kompiliert oder in einer virtuellen Maschine läuft. Der Nachteil an virtuellen Maschinen, wie sie zum Beispiel von Java benutzt werden, ist, daß man einen gewissen Overhead für den Betrieb der Maschine einbeziehen muß, was bei einer 3-D-lastigen Anwendung dazu führt, daß man relativ neue Hardware benötigt, um die Applikation flüssig betreiben zu können, oder daß man das zugrunde liegende Modell wesentlich stärker abstrahieren

[2] Kommandozeilenprogramme wie top, df, lsof, mount et cetera

[3] Die Namen der meisten Funktionen und Objekte fangen nicht von ungefähr mit NS* an.

muß.

Somit ist die Sprachauswahl durch folgende Parameter beeinflußt:

- es muß eine Hochsprache sein,
- Lauffähigkeit auf allen gängigen Plattformen,
- Lauffähigkeit auch auf älteren Systemen,
- Verwendung reiner Open Source Technologie,
- Verfügbarkeit der benötigten Bibliotheken für Grafik und Kommunikation,
- Verfügbarkeit einer leistungsfähigen IDE.

All diese Erwägungen grenzten die möglichen Sprachen schon relativ weit ein. In Frage gekommen wäre Java, welches in einer virtuellen Maschine läuft und an den Client-Rechner damit hohe Anforderungen für die 3-D-Grafik stellt; C++ mit Qt, was allerdings nicht ohne weiteres auf allen Plattformen läuft und weil QT eine nicht wirklich offene Lizenz hat; und FreePascal, welches zwar in der Entwicklung der graphischen Interfaces für MacOS X noch nicht sehr weit ist, für diese Ansprüche allerdings genügt.

Im Endeffekt fiel die Wahl auf FreePascal, da der erzeugte Code schnell, die Sprache mir bekannt und die IDE sehr leistungsfähig ist, alles unter klassischen Open-Source-Lizenzen steht und die benutzten graphischen Komponenten auch für MacOS, welches mein Entwicklersystem ist, verfügbar sind. Der Aufwand, das Programm für mehrere Betriebssysteme zu kompilieren, ist ebenfalls gering und überschaubar. Weitere Vorteile von Freepascal sind das Nichtvorhandensein von Makefiles. Der Compiler findet selbständig die Dateien, die eingebunden werden müssen. Auch die Tatsache, daß in Freepascal jede Unit ihren eigenen Namensraum hat, erleichtert die Entwicklung ebenso wie die Sauberkeit und immanente Lesbarkeit des Codes und die Geschwindigkeit des Compilers.

### 4.1.2 Programmierumgebung

Natürlich legt man sich mit der Wahl der Sprache auch auf eine beschränkte Auswahl an IDEs fest. Auch für FreePascal gibt es mehrere Möglichkeiten.

Unter Mac OS X wären da eine Integration von FreePascal in die Apple-eigene XCode-Umgebung zu nennen, das FreePascal-PlugIn für BBEdit und Lazarus, eine IDE für FreePascal, die auch in FreePascal geschrieben wurde und unter der LGPL vorliegt.

Neben den mittlerweile üblichen IDE-Funktionen wie Code-, Variablen- und Funktions-Vervollständigung und der Verwaltung der Projektdateien, der Compiler-Optionen und der benutzten Bibliotheken übernimmt Lazarus auch das Design der grafischen Bedienoberfläche und bietet die sehr komfortable Möglichkeit, Bezeichner sowie Funktions- und ähnliche Deklarationen sofort zu finden. Auch die in FreePascal gegebene Möglichkeit, für unterschiedliche Plattformen den selben Code zu verwenden und nur anders zu kompilieren, wird von Lazarus mit einem grafischen Frontend versehen und ist einfach zu benutzen.

Lazarus selbst begann als der Versuch, eine OpenSource-Variation von Delphi, Borlands mächtiger Pascal-IDE, zu schaffen, überholte aber im Funktionsumfang und besonders mit der Möglichkeit, auch für den Macintosh entwickeln zu können, sein ursprüngliches Vorbild. So ist Lazarus auch in der Lage, bereits bestehenden Delphi-Code zu kompilieren und zu bearbeiten.
Lazarus ist nicht von einer bestimmten API (Application Program Interface) abhängig, sondern benutzt für jede Zielplattform deren natives Widget-Set. Wenn das Programm zum Beispiel unter MacOS X entwickelt wurde und für die Win32-API kompiliert werden soll, linkt der Code statt zum Carbon-Interface zur Win32-API und erzeugt so ein grafisches, unter Windows lauffähiges Programm. Lazarus stellt somit eine umfangreiche RAD (Rapid Application Development)-Umgebung zur Verfügung.

Da zwar der Compiler und die IDE selbst unter der GPL, die FreePascal Component Library (FCL) und die Lazarus Component Library (LCL) aber unter der LGPL stehen, ist es möglich, mit Lazarus kommerzielle Programme zu entwickeln, ohne den Quellcode offenlegen zu müssen.

### 4.1.3 Versionsmanagement

Versionsierungswerkzeuge, auch Versionsmanagement-Software genannt, sind Programme, die es ermöglichen, Text- und Binärdateien in allen Stufen ihrer Veränderung zu bevorraten. Der große Vorteil einer solchen Dateiorganisation ist neben der vereinfachten Fehlersuche, da man Versionen, in denen eine gewisse Funktion noch funktioniert hat, sehr einfach mit der Version, in der

es nicht mehr funktioniert, vergleichen kann.

Zudem ist aber das Versionsmanagement auch eine sehr komfortable Form der Sicherungskopie, da es ohne Probleme möglich ist, das sogenannte Repository zusätzlich zur lokalen Kopie auch auf einem entfernten Server zu lagern. Wenn dann – wie während der Arbeit passiert – der Arbeitsrechner vollständig abstürzt, kann der Programmierer sich alle für die Arbeit relevanten Dateien mit allen Vorversionen vermittels eines einzelnen Befehls wieder aus dem Repository holen.

Für diese Arbeit wurde das Versionierungswerkzeug Subversion (SVN) benutzt, da es unter einer OpenSource-Lizenz frei erhältlich ist und auf nahezu allen Plattformen läuft. Ein weiterer Vorteil gegenüber anderen Versionierungssystemen, die frei erhältlich sind, wie beispielsweise dem Concurrent Version System CVS, ist die Möglichkeit, nicht nur Dateien, sondern ganze Ordnerstrukturen umzubenennen und zu verschieben. Zudem gibt es einige grafische Bedienoberflächen, die Vergleiche zwischen Versionen noch einfacher machen.

## 4.2 Modell

Wie vorher genannt, soll diese Arbeit keine möglichst getreue Rekonstruktion der Basilica Aemilia sein, sondern dient als Prototyp, als Beispielmodell für dieses Programm. Auch läßt das Modell noch einiges zu wünschen übrig: Es fehlen Texturen, die Normalen sind nicht einwandfrei, auch das Detail-Level könnte kleinteiliger sein. Für eine Demonstration der Möglichkeiten eines Bauwerksinformationssystems im Sinne eines Design- und Funktionsprototypen ist das Modell jedoch durch die Erfüllung nachgenannter Anforderungen hinreichend geeignet.

### 4.2.1 Anforderungen

Um die Auswählbarkeit einer Objektgruppe zu gewährleisten, muß das Modell in eben jene Objektgrupen zerteilt werden und jeder Teilekomplex als eine einzelne Datei im WaveFront-Object-Format, im Folgenden .obj, gespeichert werden.
Da das zu zerlegende Modell meist in einem anderen Format vorliegt, muß der Bearbeiter sich einen Arbeitsfluß aufbauen, der es erlaubt, die relevanten

Teile eines Modells komfortabel zu markieren und dann als Einzeldateien zu speichern.

### 4.2.2 Modeller

Auch wenn das Modell nicht Teil dieser Arbeit ist, so ist doch die Struktur und damit die Erstellung desselben für diese Arbeit und eventuell folgende Rekonstruktionsversuche, die diese Arbeit benutzen, wichtig.

Verschiedene Modeller weisen verschiedene Herangehensweisen an den Prozess des Modellierens auf: Architektur-CAD-Programme wie Spirit oder ArchiCAD sind auf die Bedürfnisse von Architekten ausgelegt; technische, AutoCAD zum Beispiel, auf die von Ingenieuren; und allgemeine wie zum Beispiel Blender oder Maya versuchen sich als Allrounder, die durch Plug-Ins erweiterbar sind und damit fast jedem Anspruch genügen sollen. Allerdings sollte die Wahl des passenden Modellers nicht nur, obwohl dies der Hauptentscheidungsgrund ist, von seinem Spezialgebiet abhängig gemacht werden. Weitere Kriterien sind Bekanntheit, da die Einarbeitung in ein neues CAD-Programm aufwendiger ist als bei vielen anderen Programmarten, und natürlich auch die möglichen Exportformate. Ein weiterer wichtiger Punkt ist auch die Verfügbarkeit von vorgefertigten Bauteilen. Diese können entweder als kommerzielle Bibliotheken hinzugekauft oder als frei erhältliche aus dem Netz geladen werden. Auch da gibt es wieder eine Formatproblematik: kann das Programm meiner Wahl das Format der besten Bibliothek für diesen Zweck laden?

Die naheliegendste Wahl für den Modeller für eine archäologische, architektonische Rekonstruktion ist ein CAD-Programm mit dem Schwerpunkt auf Architektur (CAAD, Computer Aided Architectural Design). Die Wahl für dieses Modell fiel auf ArchiCAD, hauptsächlich aus Gründen der Bekanntheit und Verfügbarkeit. Außerdem bot ArchiCAD auch in der Studentenversion die Möglichkeit, seine Daten als VRML zu exportieren. Ein direkter Export in das WaveFront-.obj-Format ist der kommerziellen Version vorbehalten, so mußte ein weiteres Programm benutzt werden, die VRML- in .obj-Datein zu konvertieren.

### 4.2.3 Vorgehensweise

Nach der Erstellung des Modells mit ArchiCAD war Wings3-D für die Konvertierung ob der vielen unterschiedlichen Ex- und Importformate, die dieses

Programm bietet das Programm der Wahl; außerdem war es das einzige, welches das von ArchiCAD exportierte VRML dem Zweck angemessen darstellen konnte. Wings3-D wurde im Rahmen dieses Projekts rein für den Export in das geforderte Format benutzt.

Anschließend wurde das gesamte Modell im Blender geöffnet, die einzelnen Objektgruppen zusammenhängend mit der Funktion "Select linked Vertices" ausgewählt und als Einzeldateien im .obj-Format in ein einzelnes Verzeichnis exportiert.

Dies ermöglicht ein rekursives Laden aller Dateien dieses Verzeichnisses und die automatische Erstellung einer Liste der anwählbaren Teile. Ein nächster, in diesem Prototyp noch nicht realisierter Schritt wäre die Kompression der textbasierten .obj-Dateien mit einem freien Kompressionsalgorithmius, zum Beispiel gzip, um die Dateigröße bei neuen oder erweiterten Modellen für den Benutzer gering zu halten.

## 4.3 Darstellung

Wie schon erwähnt, soll die Darstellung des Modells auf jedem unterstützten Rechnersystem nativ als OpenGL auf der Grafikkarte berechnet werden. Zu diesem Zweck wurde die FreePascal 3-D-Bibliothek Asmoday gewählt.
Die für die Darstellung benutzte Klasse `TArchMesh` ist von der grundlegenden `TAsmMesh`-Klasse abgeleitet und erweitert diese um die leicht veränderte Laderoutine aus dem Asmoday-Editor und die Methoden, den besten Betrachtungspunkt für jedes Objekt zur Verfügung zu stellen.

### 4.3.1 3-D und Plattformunabhängigkeit

Asmoday ermöglicht das direkte Ansprechen der Grafikkarte, wobei abstrahierte Objekte und Funktionen benutzt werden können, damit der Programmierer OpenGL nicht als eigene – recht maschinennahe und damit etwas unhandliche – Programmiersprache erlernen muß. Eine Kamera ist eine Kamera, egal, ob auf Windows oder Linux. Sie kann direkt benutzt werden, und muß sich nicht um die eigentliche Implemetierung auf OpenGL-Ebene kümmern.

### 4.3.2 Probleme

Bei der Benutzung von Asmoday ergaben sich einige Probleme, die aber alle durch Hilfsbereitschaft des Entwicklers und eigene Arbeit gelöst werden konnten.
Zunächst mußte Asmoday, das als reine Linux-Lösung konzipiert war, auch unter OS X funktionsfähig gemacht werden, und zwar nicht nur als X11-Komponente, was von Anfang an funktionierte, da OS X auf ein BSD 4.4-Subsystem aufsetzt, sondern als native Carbon-Anwendung. Nachdem die Bibliothek, nicht aber der Editor - da das Carbon-Widgetset noch nicht vollständig ist - unter Carbon liefen, wurde die .obj-Laderoutine des Editors an dieses Projekt angepaßt und die Unit um einige Eigenschaften erweitert.

### 4.3.3 Vorgehensweise

Als erstes galt es, einen rekursiven Ladevorgang zu implementieren, der alle Dateien eines Verzeichnisses liest und ihre Namen für die spätere Verwendung mit dem Proxy speichert. Desweiteren hat jede Bauteilgruppe eine Kameraposition, von der aus man sie am besten betrachten kann. “Am besten” heißt in diesem Zusammenhang möglichst vollständig und in ihrem Gebäudekontext. Diese Positionen werden für dieses Projekt zunächst fest im Code verankert, in einer späteren Ausbaustufe wäre es wohl sinnvoll, diese in der Datenbank zu speichern. Ferner wurden einige Debugging-Routinen benötigt, da die Behandlung von Koordinaten unter OS X etwas anders funktioniert als unter Linux oder Windows. So ist der Fensternullpunkt unter OS X der linke, oberste Punkt des Fensters mit Fensterleiste, unter Linux und Windows aber ohne. Diese Unterscheidung wurde nach ihrer Entdeckung von den Entwicklern der Lazarus-IDE in das OS X Widget-Set integriert. Nun zum eigentlichen Objekt:

ArchMesh-Listing

```
type

  { TArchMesh }

  TArchMesh = class(TAsmMesh)
  private
    FBestPos: TVector3;
    FBestView: TVector3;
    FObjName: String;
```

```
10
      procedure SetBestPos(const AValue: TVector3);
12    procedure SetBestView(const AValue: TVector3);
      procedure SetObjName(const AValue: String);
```

Die privaten Variablen und Methoden beinhalten die ideale Ansicht für jedes Objekt sowie dessen Namen.

```
13  public
      function LoadMeshFromObjFile(const Filename: string
        ): boolean;
15
      function LoadMeshFromObjStream(AStream: TStream):
        boolean;
17
      function LoadMeshFromObjStrings(const Data:
        TStrings): boolean;
```

Diese Ladefunktionen sind weitgehend aus dem Asmoday-Editor übernommen, wurden aber für dieses Projekt um die Fähigkeit des rekursiven Ladens eines Verzeichnisses erweitert.

```
      property ObjName: String read FObjName write
        SetObjName;
19
      property BestPos: TVector3 read FBestPos write
        SetBestPos;
21
      property BestView: TVector3 read FBestView write
        SetBestView;
23  end;
```

Hier wird über öffentliche Methoden das Setzen und Lesen der idealen Ansichts- und Kamerapositionen ermöglicht.

```
23 function StringToGLFloat(const Source: string; Start,
     Size: integer): GLfloat;

25 function StringToInteger(const Source: string; Start,
     Size: integer): integer;

27 function dbgs(const v: TVector3): string; overload;
```

Diese Funktionen sind für die Laderoutine nötig, da die in ASCII-Text vorliegende Datei in OpenGL-Datentypen umgewandelt werden muß. Die letzte

Funktion wandelt wiederum einen OpenGL-Datentyp in von Menschen lesbare Debugging-Informationen um. Einige der Funktionen werden wohl nach Abschluß der Arbeit in Asmoday integriert werden.

## 4.4 Datenbank

Arachne vollständig zu dokumentieren wäre eine Arbeit für sich. Hier werden nur die für die Verortung von Bauteilen im 3-D-Modell nötige Erweiterung und ihre Grundlagen angesprochen.

### 4.4.1 Anforderungen

Die Hauptanforderung ist die Übermittlung eines HTML-Tags, welches dem Objekt seinen Platz im 3-D-Modell zuweist. Dies sollte schnell und einfach zu implementieren sein, und nicht weitergehend in die übrige Struktur der Datenbank eingreifen. Außerdem sollte diese Zuordnung flexibel gehalten werden und es dem informierten Datenbank-Eingebenden nicht schwer machen, den Parameter hinzuzufügen und zu verändern.

### 4.4.2 Struktur

Die relevanten Objekte sind in Arachne als Einzelobjekte gespeichert, mit einer 1:1-Relation auf eine Tabelle, die die speziellen Charakteristika von Bauornamentik festhält. Diese Objekte wiederum werden N:M mit Bauwerken verbunden. Bei vielen Objekten ist klar, zu welchem Bauwerk sie gehören, so daß für die meisten Fälle die Beziehung 1:N ausfällt. Es gibt jedoch immer wieder Bauteile, die wegen ihrer Sturzlage, als Zweitverwendung oder aufgrund stilistischer Erwägungen zu mehreren Gebäuden passen könnten; daher die grundlegende Konzeption als N:M-Beziehung.

Zudem kann jedes Gebäude aus mehreren Bauwerksteilen bestehen. Als Bauwerksteile werden Elemente angesprochen, die einen statischen Einfluß auf das Bauwerk haben. So ist ein Geschoß, eine Galerie oder eine Porticus ein Bauwerksteil, eine Orgel in einer Kirche oder eine muslimische Minber aber ein Einzelobjekt, da dies ohne weitere bauliche Eingriffe (von Dübeln und nichttragenden Verbindungen abgesehen) aus dem Gebäude entfernt werden kann. Diese Bauwerksteile werden in einer späteren Ausbaustufe dieses Projekts noch an Bedeutung gewinnen, da man bei komplexen Gebäudestrukturen wie den Kaiservillen auf dem Palatin auch Räume und Gebäudeteile, die mit Datensätzen versehen sind, im 3-D-Modell anzeigen können möchte.

Für die hier vorliegende Arbeit ist das Einzelobjekt in seiner Verknüpfung mit dem Bauwerk maßgeblich.

Da jedes Bauwerk immer mehrere Einzelobjekte beinhalten kann, Arachne aber ob der seltenen Fälle, in denen ein Einzelobjekt mehreren Bauwerken zugeordnet werden muß, eine Kreuztabelle 'bauwerkausstattung' besitzt, wurde eben jene Kreuztabelle zur Speicherung der 3-D-Verortung benutzt. So kann bei Unstimmigkeiten, ob ein Teil jetzt nun zu diesem oder jenem Bauwerk gehört, eine Mehrfachzuordnung realisiert werden, die auch mit eventuellen weiteren 3-D-Modellen kommunizieren kann.

### 4.4.3 Vorgehensweise

In die Tabelle *'bauwerkausstattung'* wurde ein varchar-Feld mit dem Namen 'Bauteilgruppe' eingefügt. In dieses kann – händisch oder als Batch, wenn es eindeutige Kriterien in anderen Tabellen gibt – der Name der Bauteilgruppe eingetragen werden. Dieser Name ist identisch mit dem Dateinamen, der die 3-D-Objekte in der .obj-Datei beinhaltet. So sind beispielsweise alle Kapitelle des Mittelschiffs im Erdgeschoß der Aula in der Datei "AulaEGMitteKapitelle" zusammengefaßt, und bei allen in der Datenbank befindlichen Stücken, die dorthin rekonstruiert werden, steht im Feld 'Bauteilgruppe' eben jener Inhalt. Um direkt einem Kritikpunkt vorzugreifen: Sicher wäre es strukturell und semantisch richtiger, den Bauteilgruppen einfache Nummern oder sonstige, nicht Semantik tragende Codes zu geben. Wenn man sich jedoch die Menge an Bauteilen und Gebäuden anschaut, und dann überlegt, wieviele davon in absehbarer Zeit ein oder mehrere 3-D-Modelle als mögliche Rekonstruktionen erstellt und zugewiesen bekommen werden, ist es eine für die Bearbeiter einfachere Entscheidung, die aus der Praxis kommt, "sprechende" Namen für die Bauteilgruppen zu wählen, beruhend auf der aktuell bearbeiteten Rekonstruktion. Diese Namen der Bauteilgruppen sind nur für den Eingebenden sichtbar, der eigentliche Benutzer bemerkt diese nicht und für den Eingebenden wird direkt klar, wohin diese Gruppe primär gehört.

Nachdem die Tabelle *'bauwerkausstattung'* mit der *'objekt'*-Tabelle über einen Left-Join verbunden wurde, und damit die Felder dieser Tabelle auch von der *'objekt'*-Tabelle zugänglich sind, wird zunächst abgefragt, ob im Feld 'Bauteilgruppe' ein Wert steht. Wenn ja, wird dieser als unsichtbares div-Tag in den HTML-Code eingefügt:

```
if (!empty($Objekt_item['Bauteilgruppe'])){
$naviDiv = new DivWidget('object');
```

```
$naviDiv->addAttribute('style', 'visibility:hidden;');
$naviDiv->addAttribute('id', '3dDiv');
$naviDiv->addText($Objekt_item['Bauteilgruppe']);
$this->output->addBody($naviDiv);
}
```

Dieses Tag wird dann im weiteren vom Proxy erkannt und zum Hervorheben an den Viewer weitergegeben. In die andere Richtung, wenn eine Bauteilgruppe im 3-D-Fenster ausgewählt wird, wird vom Proxy eine CGI-Anfrage an die Datenbank gestellt:

```
http://arachne.uni-koeln.de/arachne2/index.php?search[
  view][kategorie]=objekt&search[view][page]=0&view[
  layout]=search_meta&view[navigation]=Kategorien&
  objekt[search][Bauteilgruppe]=NameDerBauteilgruppe
```

Dies bringt von der Datenbank eine Suchergebnismenge in der entsprechenden Suchergebnisanzeige hervor, welche dann vom Browser dargestellt werden kann.

## 4.5 Kommunikation

Der Knotenpunkt all dieser Bestrebungen ist die Kommunikation des Klientensystems mit der Onlinedatenbank. Diese schickt HTML-Seiten und akzeptiert HTML PUT als Datenanfragen über Eingabefelder und CGI als Kommandostrings.
Möchte diese Ein- und Ausgaben generieren respektive parsen, um an die erforderlichen Daten zu kommen, muß eine Komponente den Datenverkehr zwischen 3-D-Modell, Browser und Server beeinflussen können.
Es gibt insgesamt zwei Objekte in dieser Datei, `TArch3DProxyDaemon` und `TArch3DProxyThrd`, die beide von der FreePascal-Klasse `TThread` abgeleitet sind. `TThread` stellt eine Vielzahl von Methoden für Applikationen zur Verfügung, die Multithreading benötigen.

### 4.5.1 Anforderungen

Die Anforderungen an dieses Kommunikationsmodul sind

- Weiterleiten und Verändern aller Anfragen und HTML-Seiten,
- Cookie speichern,
- neue Seiten vorhalten und den Browser zum Laden selbiger bringen.

### 4.5.2 Vorgehensweise

Nach mehreren Versuchen mit unterschiedlichen Bibliotheken kristallisierten sich zwei mögliche Kandidaten heraus: die libcurl und Synapse. Für beide gibt es funktionsfähige Implementationen für Freepascal, beide beherrschen die Kommunikation über sämtliche gängigen Internetprotokolle, sind handlich und gut bedienbar. Nach mehreren Versuchen stellte sich jedoch die libcurl als eher ungeeignet heraus, da sie zwar Proxys bedienen, nicht aber ohne weiteres selbst als solcher funktionieren kann.

Synapse hingegen kann dieses. Es bedurfte zwar noch einigen Aufwandes, Synapse als Lazarus-Komponente unter MacOS X lauffähig zu machen, da MacOS X mit einer anderen Version der libc arbeitet als Linux oder Windows. Nachdem dies jedoch passiert war und alle Probleme mit Little und Big Endian ausgeräumt waren, war es kein großes Problem, einen Webserver als Proxy aufzusetzen und POST-Anfragen an den Datenbankserver zu senden. Im folgenden nun die Objektdeklaration und deren Funktionalität:

Objektdeklarationen der Proxy

```
type
2
  { TArch3DProxyDaemon }
4
  TArch3DProxyDaemon = class(TThread)
6
  private
8   FOnBrowserArchObjectChanged: TThreadMethod;
    FPort: string;
10  Sock:TTCPBlockSocket;
    ACritSec: TCriticalSection;
```

Diese Deklarationen sind für die unterste Ebene der Proxyfunktionalität zuständig. Port, Socket und die kritischen Sektionen werden initialisiert – die Veränderung des aktuellen Objekts muß als Thread-Methode vonstatten gehen, damit sich die Browser- und der Haupt(3-D)-Thread nicht gegenseitig behindern.

```
13  fBrowserArchObject: string;
    fVisualArchObject: String;
15  function GetBrowserArchObject: string;
    function GetVisualArchObject: string;
```

```
17      procedure SetBrowserArchObject(const AValue: string
          );

19      procedure SetVisualArchObject(const AValue: string)
          ;
```

Hier werden die Objektmethoden für die einzelnen Bauteilgruppen/archäologischen Objekte deklariert. Die Visual-Methoden/Funktionen sind für den Haupt-Thread, die Browser-Methoden für die Browser-Anforderungen zuständig; für beide gibt es selbstverständlich jeweils eine Set- und eine Get-Methode.

```
21    public
        constructor Create(Port: string);
23      destructor Destroy; override;
        procedure Execute; override;
```

Kontruktor, Destruktor und die eigentliche, ausführende Methode für den Proxy.

```
25
        property Port: string read FPort;
27
        property BrowserArchObject: string read
          GetBrowserArchObject write SetBrowserArchObject;
29
        property VisualArchObject: string read
          GetVisualArchObject write SetVisualArchObject;
31
        property OnBrowserArchObjectChanged: TThreadMethod
          read FOnBrowserArchObjectChanged write
          FOnBrowserArchObjectChanged;
33    end;
```

Die öffentlichen Methoden, die einzelnen archäologischen Objekte anzusprechen, und das Event, wenn sich das aktive Objekt verändert hat. Dieses Event muß als TThreadMethod deklariert werden, damit es voll multithreadingfähig ist.

```
35    { TArch3DProxyThrd }
```

```
37  TArch3DProxyThrd = class(TThread)

39  private
      FDaemon: TArch3DProxyDaemon;
41    fRegExpr: TLazRegExpression;
      Sock:TTCPBlockSocket;
```

Die Objektdeklarationen für den einzelnen Thread. Die Verwaltung der Sockets wird von Synapse übernommen, die regulären Ausdrücke, um die Daten, die durch die Proxy fließen, zu parsen, müssen auch hier thread-sicher initialisiert werden, da unterschiedliche Anforderungen eigene Threads starten.

```
43
    public
45    Headers: TStringList;
      InputData, OutputData: TMemoryStream;
```

Die öffentlichen Eigenschaften; die Header der HTML-Dokumente und der eigentliche Inhalt.

```
47
      constructor Create (hsock:tSocket; Daemon:
        TArch3DProxyDaemon);
49
      destructor Destroy; override;
51
      procedure Execute; override;
```

Auch hier: Konstruktor, Destruktor und die eigentliche Thread-Ausführung.

```
53
      function ProcessHttpRequest(Request, URI: string):
        integer;
55
      function ProcessRedirectXMLRequest(Request, URI:
        string): integer;
57
      function Local2ArachneURI(uri: string): string;
```

ProcessHttpRequest sorgt unter anderem für Fehlerbehandlung, Senden und Empfangen von CGI/HTML, liest Header und verändert Inhalte. Das ProcessRedirectXMLRequest erstellt eine einfache, temporäre XML-Datei mit

nur einem relevanten Element, aus der der Browser – durch ein später genanntes AJAX-Script – ständig erfährt, ob und was er neu laden soll. Local2ArachneURI ist eine Hilfsfunktion, um die URIs für den Browser zwischen `localhost` und `arachne.uni-koeln.de` verändern zu können.

```
59
      property Daemon: TArch3-DProxyDaemon read FDaemon;
61    property RE: TLazRegExpression read fRegExpr;
    end;
```

Die Referenzierung von Dämon und regulären Ausdrücken.

```
63
  procedure DumpLog(const Msg: String);
```

Die für das Multithreaded Debugging relevante Prozedur, welche für jeden Thread eine eigene Logdatei anlegt, und die Debugging-Informationen dieses einen Threads darin speichert.

### Exkurs: AJAX

AJAX[aja06] (Asynchronous JavaScript and XML) bezeichnet eine Technik, mit der es möglich ist, nur einzelne Teile eine HTML-Seite oder auch nur reine Nutzdaten laden zu können, ohne die gesamte Seite neu laden zu müssen. Für dieses Projekt wurde ein kleiner Teil der AJAX-Funktionalität benötigt.

AJAX-Script

```
2 var xmlHttp = false;
  /* Create a new XMLHttpRequest object to talk to the
    Web server */
4 function getItemPageData() {
        xmlHttp = false;
```

Der folgende Teil ist nötig, um Kompatibilität mit dem Internet Explorer zu gewährleisten; ohne diesen Teil funktioniert es nicht.

```
7       /*@cc_on @*/
        /*@if (@_jscript_version >= 5)
9       try {
```

```
                xmlHttp = new ActiveXObject("Msxml2.
                  XMLHTTP");
11          } catch (e) {
                try {
13              xmlHttp = new ActiveXObject("Microsoft.
                  XMLHTTP");
                } catch (e2) {
15          xmlHttp = false;
                }
17          }
            @end @*/
```

Nun werden einige Fehlervermeidungen aufgerufen und die Anforderung abgeschickt.

```
19
            if (!xmlHttp && typeof XMLHttpRequest != '
              undefined') {
21              xmlHttp = new XMLHttpRequest();
            }
23          // Build the URL to connect to
            var url = "test.xml";
25          // Open a connection to the server
            xmlHttp.open("GET", url, true);
27          // Setup a function for the server to run when
              it is done
            xmlHttp.onreadystatechange = updateItemPage;
29          // Send the request
            xmlHttp.send(null);
31  }
```

Hier beginnt die eigentliche Prozedur; wenn etwas im ersten Kind-Element steht, wird dies dem Browser als URL übergeben.

```
33  function updateItemPage() {
            if(xmlHttp.readyState == 4) {
35              response = xmlHttp.responseXML;
                if(response.getElementsByTagName('
                  urlStamp')[0].firstChild != null) {
```

```
37                              urlstamp = response.
                                  getElementsByTagName('
                                  urlStamp')[0].firstChild.data
                                  ;
                                location.href = urlstamp;
39                      } else {
                            window.setTimeout("getItemPageData()
                                ", 1000);
41                      }
                }
43 }
```

Einmal pro Sekunde wird überprüft, ob es eine neue URL gibt.

```
45 window.setTimeout("getItemPageData()", 1000)
```

Das XML-Dokument hat folgende Form:

```
1
<?xml version ="1.0"?>
3 <proxyCheck>
        <urlStamp>NewUrl</urlStamp>
5 </proxyCheck>
```

Dieser Mechanismus reicht aus, um einen Browser mit aktiviertem Javascript dazu zu bewegen, eine neue URL zu laden und damit eine neue Suchergebnismenge aufzurufen, die der Bauteilgruppe entspricht.

### Exkurs: Multithreading

Multithreading[TW97] bezeichnet ein Prinzip, welches es ermöglicht, im selben Prozeß mehrere Aufgaben des Prozesses gleichzeitig abzuarbeiten; nicht wirklich simultan, sondern durch Hin- und Herspringen zwischen den einzelnen Aufgaben des Prozesses. Ein einfaches Beispiel: Ein Brower fordert eine HTML-Seite an, die aus mehreren Dokumenten wie Bildern und HTML-Dateien besteht. Nun gibt es für den Browser zwei Möglichkeiten: Entweder lädt er die Dokumente seriell, das heißt, er fordert vom Server die erste Datei der Seite an, wartet, bis diese da ist, fordert die nächste an und so weiter. Dort entsteht das erste Problem: Was, wenn die erste Datei auf einem Server liegt, der sehr lange Latenzzeiten hat? Der User muß warten, bis der Server die gewünschte Datei geliefert hat und die nächste geladen werden kann, obwohl

die Netzwerkschnittstelle und andere Ressourcen des Computers bei weitem nicht ausgelastet sind. Die passende Antwort ist Multithreading: Statt eines einzelnen werden soviele Threads, wie das Dokument Dateien hat, gestartet, und wenn das Laden einer Datei wesentlich länger dauert als das einer anderen, wird die schnellere Datei gleichzeitig geladen. Diese Threads sind eigenständige Instanzen desselben Prozesses; sie greifen gegebenfalls auf die gleichen globalen Variablen zu und arbeiten denselben Code ab. Dabei hängt die Art und Weise, wie mit Threads umgegangen wird, stark von der verwendeten Prozessor- und Betriebssystemarchitektur ab.

Es gibt zwei grundlegend unterschiedliche Verfahren: das kooperative und das präemptive Multitasking. Beim kooperativen Multitasking muß jeder Thread selber die Ressourcen, die er benutzt, wieder freigeben. Beim präemptiven Multitasking teilt das Betriebssystem als übergeordnete Instanz den einzelnen Prozessen die Rechenzeit zu. Die meisten modernen Betriebssysteme benutzen das präemptive Multitasking, da es auf der einen Seite für die Programmierer weniger Aufwand bedeutet, sich nicht in extensio um die Thread-Verwaltung kümmern zu müssen, und auf der anderen Seite das Betriebssystem dann die volle Kontrolle über die Ressourcen behält. Eine Ausnahme sind Anwendungen, bei denen die Latenzzeit möglichst gering sein muß, so etwa Audiobearbeitung oder kritische Anwendungen des Messen-Steuern-Regelns. Als Beispiel sei hier das Multithreading unter Unix-Systemen genannt, doch auch hier gibt es Unterschiede; einige Unix-Systeme arbeiten mit der Mach C-Threads-Bibliothek, andere mit POSIX P-Threads[But97], wobei es auch zahlreiche Kombinationen aus beidem gibt. P-Threads arbeiten mit dem Mutex genannten Datentyp, um Blockaden durch andere Prozesse und Threads zu vermeiden. Genau dieses Modell führte dazu, daß der Code unter Linux einwandfrei funktionierte, unter OS X allerdings Fehler erzeugte, da die Threads dort eine Mischung aus C- und P-Threads darstellen, die vermittels einer 'Funnel' genannten Technik zusammengebracht wird.

So auch im Falle dieses Projekts. Die erste Möglichkeit, solche Fehler zu beheben ist, die Verwaltung von Threads durch den Compiler so zu verändern, daß sich ein Thread, ungeachtet der genauen Implementierung immer gleich verhält. Dies führt – außer, daß es sehr aufwendig gewesen wäre – auch zu Performanzproblemen, da nicht die unteren Betriebssystemfunktionen für die Thread-Verwaltung benutzt werden würden, sondern eine prozeßinterne Thread-Verwaltung als einzelner Thread laufen würde. Die zweite Möglichkeit ist, das Multithreading auf dem verwendeten Betriebssystem für den verwendeten Compiler zu verbessern und die Routinen zu finden, die das

fehlerfreie Ablaufen verhindern.

Wenn nun Fehler auftauchen und man mit der Ausgabe von Variablenwerten an bestimmten Programmstellen den Fehler finden möchte, gibt es auch hier mehrere Möglichkeiten, dies zu tun[DSK]. Ein Ansatz, einen Fehler im Multithreading zu finden, ist, für jeden Thread ein Logfile zu erzeugen, in dem man die für diesen Thread relevanten Meldungen speichert. Auf dieser Basis kann man dann die Zahl der Threads und die darin vorgekommenden Fehler finden. Dies erwies sich auch in diesem Projekt als sehr praktikabel, und nach einiger Zeit zeigte sich, daß, was schnell zu beheben war, die Bibliothek für die regulären Ausdrücke nicht thread-sicher und die 'Synchronize'-Funktion, mit deren Hilfe ein arbeitender Thread den Haupt-Thread aufwecken kann, noch nicht ausreichend in FreePascal implementiert war. Hier zeigte sich eine Stärke der Open-Source-Bewegung: Innerhalb weniger Tage hatte ein Compiler- und IDE-Entwickler diese Funktion implementiert.

## 4.6 Hauptprogramm

Das Hauptprogramm kümmert sich neben dem Koordinierungsaufwand für die Kommunikation zwischen Proxy und 3-D-Applikation auch um das Event-Handling, die Fenstersteuerung und die Navigation in der 3-D-Umgebung. Das Hauptobjekt dieses Programmteils ist das `TForm1`, eine abgeleitete Klasse der Lazarus-Klasse `TForm`, die fast alle Methoden für die Verwaltung eines grafischen Programmfensters zur Verfügung stellt.

### 4.6.1 Anforderungen

Neben der grundlegenden Koordination und Navigation soll das Hauptprogramm sich auch um den "Magic Carpet Ride" kümmern, der den Benutzer in einer fließenden Bewegung an die ideale Betrachtungsposition befördert. Außerdem sind hier zusätzliche OpenGL-Kontrollen implementiert, insbesondere Funktionen wie das Abfangen von Tastaturereignissen und Mausbewegungen. Auch nicht im Modell beinhaltete Objekte werden im Hauptprogramm initialisiert. Darunter fallen der Boden, die Kamera und die Beleuchtung. Gewisse Parameter wie die Art des Renderings[4] und Farbe der einzelnen Objekte werden, so sie nicht im Objekt gespeichert sind, auch hier gesetzt und verwaltet.

[4] Asmoday bietet verschiedene Optionen für Parameter wie Beleuchtung, Schatten, Texturen, Normalen oder einen Wireframe-Modus an.

### 4.6.2 Realisierung

```
type
  TMoveMode = (mmHead, mmSelect);
```

Zur Zeit gibt es zwei Bewegungsmodi: einen klassischen Erkundungsmodus im Stil eines First-Person-Shooters, und einen "Stehen bleiben und anklicken"-Modus. Es bleibt durch Benutzerrückmeldung zu evaluieren, ob diese Bewegungsmodi noch geändert werden müssen.

```
  { TForm1 }

  TForm1 = class(TForm)
    OpenGLControl1: TOpenGLControl;

    procedure FormCreate(Sender: TObject);
    procedure Form1Idle(Sender: TObject; var Done:
      Boolean);
    procedure FormDestroy(Sender: TObject);
```

Hier wird das Fenster deklariert; Konstruktor, Destruktor und ein OnIdle-Ereignis.

```

    procedure FormKeyDown(Sender: TObject; var Key:
      Word; Shift: TShiftState);

    procedure FormKeyPress(Sender: TObject; var Key:
      char);

    procedure FormKeyUp(Sender: TObject; var Key: Word;
       Shift: TShiftState);

    procedure OpenGLControl1KeyDown(Sender: TObject;
      var Key: Word;
       Shift: TShiftState);

    procedure OpenGLControl1MouseDown(Sender: TOBject;
      Button: TMouseButton;
```

```
        Shift: TShiftState; X, Y: Integer);

    procedure OpenGLControl1MouseMove(Sender: TObject;
      Shift: TShiftState; X,
      Y: Integer);

    procedure OpenGLControl1MouseWheel(Sender: TObject;
       Shift: TShiftState;
      WheelDelta: Integer; MousePos: TPoint; var
        Handled: Boolean);
```

Dies sind die Maus- und Tastaturereignisse; sie müssen für das Fenster selbst und den OpenGL-Kontext getrennt behandelt werden.

```
    procedure OpenGLControl1Paint(Sender: TObject);
    procedure OpenGLControl1Resize(Sender: TObject);
```

Die eigentliche OpenGL-Zeichenroutine und das Größenveränderungsereignis – dieses ist allerdings noch experimentell und wird nicht benutzt.

```
  private
    FMaxPositionAcceleration: double;
    FMaxViewAcceleration: double;
    Flying: Boolean;
    FlightDestinationPosition: TVector3;
    FlightDestinationView: TVector3;
    FlightVelocityPosition: TVector3;
    FlightVelocityView: TVector3;
```

Die privaten Parameter für das “Fliegen” zu den idealen Betrachtungspositionen.

```
    FProxy: TArch3–DProxyDaemon;
    light: TAsmPositionalLight;
    light2: TAsmPositionalLight;
    FPCamera: TAsmFirstPersonCamera;
    SecondCam: TAsmRotationCamera;
    Scene: TAsmScene;
```

```
    floor: TAsmObject;
51  sphere: TAsmObject;
    initialized: boolean;
```

Der Proxy und die in der Szene zu initialisierenden Objekte wie Kamera, Licht und Boden.

```
53
    MoveMode: TMoveMode;
55  lastX, lastY: integer;
    ListOfModels: TFPList;
57  SelectableObjects: TFPList; //list of TAsmObjects
```

Einerseits die Liste aller Objekte/Modelle in der Szene, andererseits die der auswählbaren Bauteilgruppen. MoveMode ist der zu Beginn genannte Ansatz, auf der einen Seite zu erkunden, auf der anderen Bauteilgruppen anzuwählen und die Szene ruhen zu lassen.

```
59  LastRenderTime: TDateTime;
    DTime: TDateTime;
61  FTime: TDateTime;
    FTimeCount: Integer;
```

Für den Zähler der Bilder pro Sekunde und das Hingelangen zum idealen Betrachtungspunkt nötige Werte.

```
63
    procedure LoadObjectsFromDir (DirectoryName :
      String);
65
    procedure LoadObjFile (Filename : String);
67  procedure CreateElementPositions;
```

Die Aufrufe der Laderoutinen aus ArchMesh und die Positionierung der Objekte – da OpenGL mit einem anderen Koordinatensystem arbeitet als die meisten Modeller, muß das Modell bei der Initialisierung um 90 gedreht werden. Die hier erzeugten Objekte wie Lampen und ähnliches müssen natürlich auch positioniert werden.

```
69  procedure OnBrowserArchObjectChanged;
```

Diese Prozedur behandelt alle Wechselereignisse von Bauteilgruppen nach der Auswahl im 3-D-Fenster oder einem Ergebnis im Browser.

```
71      procedure SetMaxPositionAcceleration(const AValue:
          double);

73      procedure SetMaxViewAcceleration(const AValue:
          double);
```

Die Set-Methoden für die höchste Geschwindigkeit und Beschleunigung beim Blickpunktwechsel.

```
75    public
        procedure InitWorld;
77      procedure PaintScene;
```

Diese Prozeduren sprechen für sich: InitWorld stellt alle Objekte zusammen und initialisiert das OpenGL, PaintScene wird bei jeder Änderung der 3-D-Ansicht aufgerufen.

```
79      procedure HandleKeydown(var Key: Word; Shift:
          TShiftState);

81      procedure Zoom(Zstep: glfloat);
        procedure MoveVert(VertStep: glfloat);
83      procedure MoveHonz(HonzStep: glfloat);
        procedure Turn(angle: GLfloat);
85      procedure MoveAlongPlane(speed: GLfloat);
```

Hier werden die Tastenereignisse abgefangen, an die privaten Methoden weitergegeben und, wenn es sich um Bewegungsereignisse handelt, ausgeführt.

```
87      procedure CreateRayFromXY(X, Y: GLdouble; var
          RayOrigin, Ray: TVector3);

89      function FindSelectableObject(const AName: String):
          TAsmObject;
```

```
91      procedure SelectObject(X, Y: integer);
        procedure SelectObject(AnObject: TAsmObject);
```

Diese vier Methoden sind für die Anklickbarkeit von Bauteilgruppen zuständig. Zuerst wird aus der zweidimensionalen Mauskoordinate der dreidimensionale Richtungsvektor bestimmt, anschließend wird anhand der Bounding-Spheres grob festgestellt, ob und was alles vom Richtungsvektor geschnitten wird, und zuletzt werden diese Objekte in Reihenfolge gebracht und bei dem vordersten beginnend geschaut, ob es eine Vektor/Polygon-Kollision gibt. Ist dies der Fall, wird das Objekt ausgewählt.

```
93
        property Proxy: TArch3–DProxyDaemon read FProxy;
```

Hier wird der Proxy als Parameter übergeben.

```
95
        procedure AnimateFlight(Seconds: double);
97
        procedure SetFlightDestination(const DestPos,
          DestView: TVector3);
99
        procedure SetFlightDestination(const ObjName:
          string);
101
        property MaxPositionAcceleration: double read
          FMaxPositionAcceleration write
          SetMaxPositionAcceleration;
103
        property MaxViewAcceleration: double read
          FMaxViewAcceleration write SetMaxViewAcceleration
          ;
```

Hier werden die Werte für den “MagicCarpet Ride” gesetzt und selbiger ausgeführt.

## 4.7 Dateien

Der gesamte Hauptprogrammcode befindet sich in der Datei `./code/ArchDB3D/simpleform.pp`,

die Laderoutine und das TArchMesh-Objekt unter
`./code/ArchDB3D/ArchMesh.pas`
und die Proxy-Bibliothek unter
`./code/ArchProxy/arch3ddbproxy.pas`.
Die einzelnen Objektdateien finden sich im Verzeichnis
`./code/BAEinzeln/`.
Die ausführbare Binärdatei wird von Lazarus in das Verzeichnis mit dem Hauptprogrammcode gelegt. Die für das Nachladen neuer Seiten zuständige JavaScript-Datei liegt auf dem Datenbankserver; eine Kopie in
`./code/javascript/ArchDB.js`.

## 4.8 Benutzung

Benötigt wird ein Account bei den Arachne-Datenbanken und eine Grafikkarte, die mindestens OpenGL 1.2 unterstützt.
Nach dem Starten des Programms in der Browser-Adresszeile
`http://localhost:16000/arachne2/`
eingeben und die Return-Taste drücken. Benutzername und Passwort für die Arachnedatenbanken eingeben.
Navigation im 3-D-Fenster:

- `W`: vorwärts
- `S`: rückwärts
- `A`: links
- `D`: rechts
- `Shift`: bei gleichzeitig gedrückter Shift-Taste wird die Bewegung schneller
- `Space`: wechselt zwischen Erkundungs- und Auswahlmodus
- `1`: Position Mitte der Aula, Erdgeschoß
- `2`: Position Galerie der Aula, Obergeschoß
- `3`: Position Frontalansicht der Basilica
- Mauszeiger: Im Erkundungsmodus Bewegen der Sicht, im Auswahlmodus zur Auswahl
- `ESC`: Beenden des Programms (vorher bitte bei Arachne abmelden)

Objekte, zu denen in jedem Fall Datensätze vorhanden sind, sind die Säulen im Inneren der Basilica. Datensätze, zu denen in jedem Fall Objekte vorhanden sind, sind zum Beispiel 123001,123006 und 123410. Da bei einer Suchergebnisanzeige nach der Markierung einer Bauteilgruppe alle dieser Bauteilgruppe zugeordneten Einzelobjekte angezeigt werden, kann man nachfolgend durch Anklicken eines einzelnen Datensatzes die Bewegung zum besten Betrachtungspunkt hin beobachten. Bei einer Ergebnismenge gibt es diese nicht, da dort mehrere Bauteilgruppen in einer Anzeige vorkommen können. Deswegen wird die Bewegung nur bei der Auswahl von Einzeldatensätzen vollzogen.

# Kapitel 5

# Fazit und Ausblick

Diese Arbeit hat gezeigt, daß es mit überschaubarem Aufwand möglich ist, ein dreidimensionales Gebäudemodell mit einer bestehenden, internetbasierten Forschungsdatenbank zu verbinden und so dem Wissenschaftler und dem interessierten Laien einen wesentlich anschaulicheren Raumeindruck eines nicht mehr vorhandenen Gebäudes zu vermitteln, als dies mit einer zeichnerischen Rekonstruktion möglich wäre. Zudem bietet dieses Projekt die Möglichkeit, sich die Grundlage für genau diese Rekonstruktion im Kontext visualisieren zu lassen und dynamisch für andere Modelle anzupassen.
Auch wurde am Beispiel der Basilica Aemilia deutlich, daß es Gebäudesituationen gibt, in denen eine Fülle von Fakten nur mühevoll zu einer schlüssigen Rekonstruktion zusammengebracht werden können. Selbst eine äußerst akribische Rekonstruktion muß für andere Wissenschaftler nachvollziehber sein, und wenn sich die Materialbasis so schwierig darstellt, wie dies bei der Basilica Aemilia der Fall ist, ist eine Visualisierung des Gebäudes mit direkt zugänglicher Materialbasis ein wichtiges Werkzeug, quellenkritisch mit diesem Vorschlag umgehen zu können.
Desweiteren ermöglicht die immanente Erweiterbarkeit des Projekts anderen Wissenschaftlern, mit überschaubarem Aufwand auch ihre Ergebnisse so präsentieren zu können. Dies führt zu einer größeren Nachvollziehbarkeit und Transparenz des Ergebnisses und zu einer leichter zugänglichen Diskussionsgrundlage.
Ein weiteres, erreichtes Ziel ist die Portabilität der Applikation. Kein größeres Betriebssystem wird ausgespart, es ist – auch, wenn das Programm noch stark erweitert werden sollte – keine “Windows only”-Anwendung. Das Zusammenspiel mit dem Browser als bestehendem Datenbankinterface hält zudem den Einarbeitungsaufwand so gering wie möglich, und die wenigen Bedienelemente des Viewers sind schnell zu erlernen.
Diese Applikation ist zudem äußerst flexibel in Hinsicht darauf, daß sie oh-

ne großen Anpassungsaufwand auf praktisch jede Apache - MySQL - PHP-Datenbank aufgesetzt werden kann, ohne den Datenbankcode in größerem Umfang verändern zu müssen.
Die Probleme, die es bei der technischen Umsetzung mit OpenSource- Komponenten wie FreePascal und Synapse gab, wurden von den Entwicklern jener Komponenten prompt und kompetent gelöst.

Zum Ausblick läßt sich sagen, daß es wünschenswert wäre, wenn mehr Modelle am Computer erstellt und die Ergebnisse der Bauaufnahme direkt elektronisch erfasst würden. Das Programm selbst wäre dann um die Möglichkeit zu erweitern, mehrere Modelle nebeneinander zu vergleichen und sich die unterschiedlichen Verortungen von Bauteilen gleichzeitig anzeigen zu lassen. Auch die Speicherung weiterer Daten, die zur Zeit noch im Programmcode fest verankert sind, wie zum Beispiel die besten Betrachtungspositionen, in der Datenbank wäre eine sicher wünschenswerte Erweiterung.
Im weiteren Verlauf ebenfalls zu implementieren wäre die Möglichkeit, auch die Eingabe der Bauteilverortung über das 3D-Interface vorzunehmen, ebenso wie die Separierung einzelner Bauteilgruppen innerhalb eines Modells. Eine solche Editor-Funktionalität würde auch dem Laien oder Studenten eine gute Möglichkeit bieten, sich mit dem Prozeß der archäologischen Rekonstruktion auseinanderzusetzen, und dem eingebenden Archäologen ein bedeutend komfortableres Datenbankinterface für die Zuordnung von Bauteilen zu Gebäuden zur Verfügung stellen. In noch weiterer Ferne, aber als denkbare Verbesserung, läge eine Einbettung der Browserkomponente direkt in das 3-D-Fenster. Die Komponente würde nur bei Datenbanktätigkeit in den Vordergrund rücken und könnte vom Benutzer frei in der 3-D-Welt platziert werden.

Abschließend läßt sich sagen, daß dreidimensionale Komponenten zur Wissenserschließung und -organisation machbar und wünschenswert sind. Sie stellen eine sinnvolle Erweiterung zu klassischen Datenbankoberflächen dar und sind in der Lage, in einem dynamisch veränderbaren Bild schneller mehr Wissen über räumliche Zusammenhänge zu vermitteln als ein Text oder statisches Bild in der Lage wäre.

# Literaturverzeichnis


[AC06] Apple Computer, Inc.: *Apple Human Interface Guidelines*. Webrelease, 06 2006. http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/index.html.

[aja06] *Ajax (Programmierung)*, 8 2006. http://de.wikipedia.org/wiki/Ajax_%28Programmierung%29.

[arc06a] *ArchiFM*, 11. Juli 2006. http://www.graphisoft-rheinmain.de/index.php?option=com_content&task=view&id=58&Itemid=30.

[arc06b] *Graphisoft ArchiCAD*, 13. Juli 2006. http://www.graphisoft.de/.

[Aut06] Autodesk: *AutoCAD DXF Reference*, 3. August 2006. http://usa.autodesk.com/adsk/servlet/item?siteID=123112&id=5129239.

[Bau77] Bauer, H.: *Kaiserfora und Ianustempel*, 1977.

[Bau88] Bauer, H.: *Augustus und die verlorene Republik*, Kapitel Basilica Aemilia, Seite 200 ff. Ausstellungskatalog Berlin, 1988.

[Bau93] Bauer, H.: *Basilica Paul(l)i*, 1993.

[Bau96] Bauer, F. A.: *Stadt, Platz und Denkmal in der Spatantike*. 1996.

[Ber65] Bergener, A.: *Die führende Senatorenschicht im frühen Prinzipat*. 1965.

[ble06] *Blender 3D*, 07 2006. http://blender.org/.

[But97] Butenhof, David R: *Programming with POSIX Threads*. Addison-Wesley, 1997.

[cae] *CIL VI 36908.*

[caf06] *Facility Management*, 07 2006. http://www.mp-gruppe.de/jsp/epctrl.jsp?pri=mps&cat=mps000001&mod=mps000010&pEvent=include&pValue=addons/survey-cafm/cafm.jsp.

[Car48] Carettoni, G.: *Esplorazione nella basilica Emilia.* 1948.

[DB01] D. Bowman, E. Kruijff, J. LaViola Jr. I. Poupyrev: *An Introduction to 3-D User Interface Design*, 2001.

[Deu06a] Deutsches Archäologisches Institut: *Libanon, Baalbek*, 3. August 2006. http://www.dainst.org/index_2951_de.html.

[Deu06b] Deutsches Archäologisches Institut: *Rom, Kaiserpalast*, 3. August 2006. http://www.dainst.org/index_3286_de.html.

[DSK] Dassen, J.H.M. und I.G. Sprinkhuizen-Kuyper: *Debugging C and C++ code in a Unix environment.* http://oopweb.com/CPP/Documents/DebugCPP/VolumeFrames.html.

[Egg74] Egger, H.: *Codex Escurialensis. Ein Skizzenbuch aus der Werkstatt Domenico Ghirlandaios.*, 1974.

[For06] Forschungsarchiv für Antike Plastik: *Arachne-Datenbanken*, 3. August 2006. http://arachne.uni-koeln.de/.

[Geb06] Gebauer, Lukas: *Synapse - asynchronous TCP/IP-Library*, 12. Juli 2006. http://synapse.ararat.cz/.

[Ger06] Gervasio, Alejandro: *Building Object-Oriented Database Interfaces in PHP.* DevShed, 3. August 2006. http://www.devshed.com/c/a/PHP/.

[Gra06] Grabowski, Reimar: *Asmoday*, 12. Juli 2006. http://asmoday.sourceforge.net/.

[Gro10] Groag, E.: *Fulvius.* Realencyclopädie der classischen Altertumswissenschaften, VII(1):265 ff., 1910.

[Hen] Henze, Frank; Brasse, Christiane: *Ein modulares Informationssystem für Archäologie und Bauforschung.* http://www.dainst.org/index_6410_de.html, 13. Juli 2006.

[HS97] Helmut Saurer, Franz-Josef Behr: *Geographische Informationssysteme. Eine Einführung.* 1997.

[Jr.79] Jr., L. Richardson: *Basilica Fulvia, Modo Aemilia*. In: Moore, G. Kopcke M. B. (Herausgeber): *Studies in classical Art and Archeology*, Seite 209 ff., 1979.

[Kle93] Klebs, E.: *Aemilius*. Realencyclopädie der classischen Altertumswissenschaften, I(1):552 f., 1893.

[Kol95] Kolb, F.: *Rom. Die Geschichte der Stadt in der Antike*. 1995.

[lat] *CIL VI, 36962*.

[Lip05] Lipps, Johannes: *Die Kapitelle der Basilica Aemilia*. Diplomarbeit, Universität zu Köln, 2005.

[MF99] Mathea-Förtsch, M.: *Römische Rankenpfeiler und -pilaster*. 1999.

[Nor49] Nordh, A.: *Libellus de Regionibus urbis Romae*. 1949.

[obj06] *The .obj Reference*, 3. August 2006. http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj/.

[sta] *CIL VI, 1658a*.

[TW97] Tanenbaum, Andrew S. und Albert S. Woodhull: *Operating systems (2nd ed.): Design and Implementation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[win06] *Wings3D*, 07 2006. http://www.wings3d.com/.