Praktikumsbericht

T-Systems

15. Juli – 15.Oktober 2005

Mirko Dietrich

23. Juli 2006

Inhaltsverzeichnis

1 Der Arbeitsplatz bei T-Systems

1.1 Das Unternehmen T-Systems

1.1.1 Die Rolle in der Deutschen Telekom

Das Unternehmen *T-Systems* ist eine 100-prozentige Tochter der *Deutschen Telekom AG*. Als Systemhaus innerhalb der Telekom betreut sie die Geschäftskunden und kümmert sich außerdem um Forschung und Entwicklung. Weiterhin gibt es innerhalb des *Telekomkonzerns* noch die Geschäftsfelder Breitband/Festnetz *(T-Com)* und Mobilfunk *(T-Mobile)*, die sich im Gegensatz zur *T-Systems* speziell an Endkunden richtet. Mit 52.000 Mitarbeiter in 20 Ländern erwirtschaftet die T-Systems einen Umsatz von 10,5 Milliarden Euro.

1.1.2 Aufgabenfelder

Das größte Betätigungsfeld bildet das *ICT-Outsourcing*. Hier werden anderen Unternehmen Leistungen in der *Informations- und Telekommunikationstechnik* angeboten. Diese können sich dadurch auf ihre speziellen Aufgaben konzentrieren und trotzdem flexibel bleiben. Außerdem ist die *Projektbetreuung* ein wichtiges Geschäftsfeld von T-Systems. Sie optimiert die Prozesse der Unternehmen und hilft so Kosten zu senken. Im Idealfall legt der Kunde dabei T-Systems einen Anforderungskatalog vor, T-Systems stellt dann einen Projektplan auf, beschafft die Technik, installiert sie schlüsselfertig, schult die späteren Benutzer und ist für den kurzfristigen Service im Falle von Ausfällen zuständig. Bekannte Projekte der T-Systems sind beispielsweise die Software für das *Arbeitslosengeld II* und das Mautsystem von *Toll Collect*.

1.1.3 Die Geschichte des Unternehmens

Früher hatte die *Deutsche Telekom* mehrere Systemhäuser: Die *DeTeSystem* sowie die *T-Nova*. Außerdem gab es einen konzerninternen IT-Dienstleister *(DeTeCSM)*. T-Systems entstand durch die Verschmelzung der genannten Tochterunternehmen und dem von *DaimlerChrysler* zugekauften Unternehmen *debis Systemhaus*.

1.2 Die Abteilung Digital Engineering Solutions

Die Abteilung *Digital Engineering Solutions* ist innerhalb von T-Systems auf *CAE*[1] spezialisiert und arbeitet unter anderem an der Weiterentwicklung der Software Medina — einem Pre- und Postprozessor für Finite-Elemente. Dies ist ein Programm, dass seine Ursprünge noch vor 1985 hat. Damals wurde es von der Entwicklungsabteilung bei *Daimler-Benz* ins Leben gerufen. Später kümmerte sich das zu Daimler-Benz gehörende debis Systemhaus um die Pflege und Erweiterung der Software. Nachdem Daimler-Benz beschloss debis abzustoßen und die Deutsche Telekom das Unternehmen übernahm

[1] Computer Aided Engineering: computergestützte Konstruktion

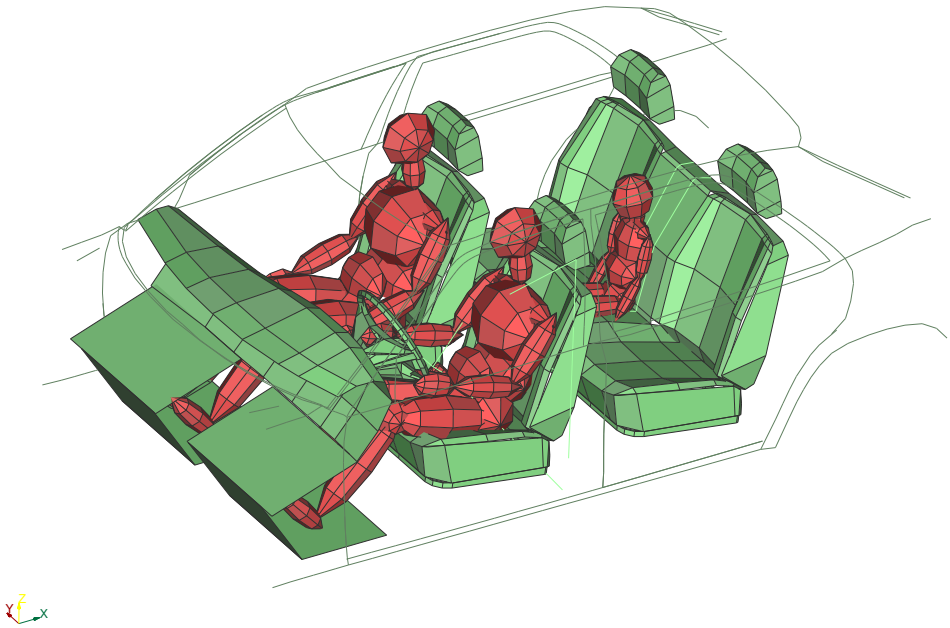

Abbildung 1: Crashtestmodell

wurde die Entwicklung innerhalb von T-Systems fortgesetzt. Noch heute ist das Unternehmen DaimlerChrysler der Hauptkunde von Medina. Die Software gehört zu den Marktführern in dieser Sparte und besitzt in Europa eine außerordentliche Marktdurchdringung. Praktisch jeder deutsche und die Mehrzahl der europäischen Automobilhersteller besitzen Medinalizenzen. In den USA ist das Produkt nur sporadisch vertreten. Dort hat sich die Software *Hypermesh* in diesem Marktsektor etabliert.

Im Automobilbau müssen kostenspielige Tests durchgeführt werden bevor Fahrzeuge in Serienproduktion gefertigt werden können. Es wird versucht durch Computersimulation Einsparungen zu erzielen. Realitätsnahe Ergebnisse können oft auch ohne reale Versuche produziert werden. So werden u.a. *Windkanal-* und *Crashtests* (siehe Abbildung 1), aber auch *Spannungs-* und *Belastungssimulationen*, sowie *thermische Berechnungen* durchgeführt. Das in *Fortran* und *C/C++* geschriebene Programm Medina übernimmt gewisse Funktionen innerhalb eines Simulationsprozesses. Werden Modelle aus der Konstruktionsabteilung in Form von *Geometrien* geliefert, so können sie mit Hilfe von Medina vernetzt werden. Dadurch entsteht ein Modell aus *Finiten Elementen*, dass sich zur Berechnung eignet. Das Programm besitzt Schnittstellen zu speziellen *Solvern*, wie *PAMCRASH* oder *ABAQUS*. Sie sind in der Lage auf der Basis finiter Elemente Berechnungen anzustellen, die wiederum als Ergebnisse in Medina importiert werden können, um dann visualisiert, ausgedruckt oder als Animation abgespeichert zu werden.

In der Abteilung sind Rechnerarchitekturen aller Art vertreten. Medina wurde ursprünglich auf einem Großrechner entwickelt. Als sich später die *Workstations* verbreiteten, wurde Medina zu einer Einzelplatzsoftware. Das Betriebssystem auf dem sich das

Programm ursprünglich „heimisch“ fühlte ist *Unix* und zwar in unterschiedlichster Couleur: *HP-UX*, *GNU/Linux*, *AIX*, *Irix*, *Solaris*. In den späten 90er-Jahren, als der *PC* die *Workstations* im Bereich der Performance überholten, wurde eine Portierung nach *Microsoft Windows* vollzogen. Doch auch heute noch sind die *Unixderivate* die Systeme, auf denen die CAE-Applikation hauptsächlich eingesetzt wird.

2 Die Aufgabe

2.1 Das Problem des Hidden Surface Removal

Hidden Surface Removal[2] ist ein in der Computergrafik immer wiederkehrendes Problem. Verdeckte Flächen dürfen bei der Anzeige am Bildschirm oder beim Ausdruck nicht angezeigt, sondern müssen von vordergründigen Objekten verdeckt werden. Dies mag aus Sicht der „realen Welt“ sinnlos erscheinen, da Materie in der Regel undurchsichtig ist, doch in der Computergrafik ist dies ein ernst zunehmendes Problem, welches in der Wissenschaft bereits umfangreich behandelt wurde. In der einschlägigen Literatur wird prinzipiell zwischen zwei verschiedenen Klassen von Algorithmen unterschieden, die diese Problemstellung lösen: *bildpräzise* und *objektpräzise Verfahren*[3]. Unumgänglich ist es dabei eine Entscheidung zu treffen, welche Elemente gezeichnet werden und welche nicht. Und in diesem Punkt unterscheiden sich die Verfahren bereits. Die ersteren entscheiden auf der Basis von Unterelementen des Bildes, z.B. Pixeln. So wird für den sogenannten *Z-Buffer-Algorithmus* für jeden einzelnen Pixel entschieden. Im Gegensatz dazu trifft die Gruppe der objektpräzisen Algorithmen Entscheidungen für jedes einzelne Objekt. Daraus ergeben sich bereits verschiedene Einschränkungen und Besonderheiten. Beispielsweise können pixelbasierte Verfahren nicht bei *vektororientierten Darstellungen* verwendet werden.

2.2 Der PostScript-Export bei Medina

Ziel des bearbeiteten Projektes war es, einen Algorithmus für das Problem des *Hidden Surface Removal* zu konzipieren und implementieren. Beim *FE Pre- und Postprozessor Medina*, der von der Abteilung *Digital Engineering Solutions* entwickelt wird, gibt es verschiedene Arten der Grafikausgabe. Die drei wichtigsten sind in Abbildung 2 dargestellt.

Es dürfen beim Export nach *PostScript*[4] keine verdeckte Flächen und Linien angezeigt werden. Hier können nur *objektpräzise Algorithmen* angewendet werden, da es sich um *Vektorgrafik* handelt und in dieser Stufe noch keine *Rasterung*, also Darstellung unter Verwendung von Pixeln stattfindet. Bei der Ausgabe auf dem Monitor übernimmt der, in der Grafikkarte auf Hardwarebasis implementierte *Z-Buffer* diese Aufgabe. Bei der Vek-

[2] zu deutsch: Entfernung verdeckter Flächen, Sichtbarkeitsproblem

[3] Allerdings gibt es auch hybride Verfahren, die Techniken beider Ansätze nutzen.

[4] Eine von Adobe entwickelte Scriptsprache, die hauptsächlich verwendet wird um Drucker und andere Ausgabegeräte anzusteuern.

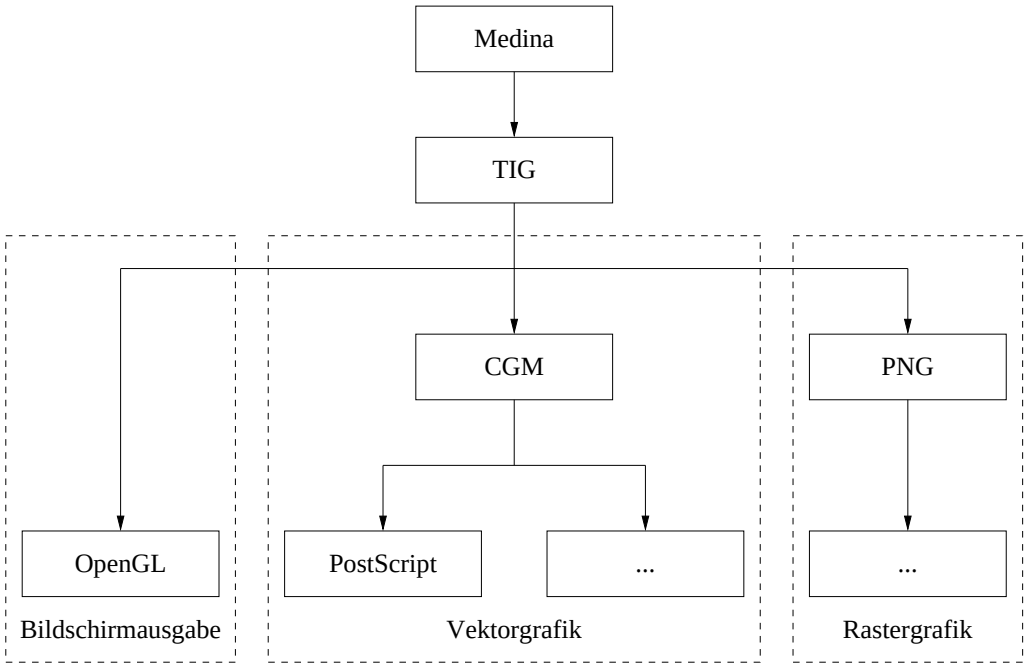

Abbildung 2: Schematische Darstellung der Grafikausgabe bei Medina

torausgabe wurde die Grafik bisher sukzessive von hinten nach vorne aufgebaut, so dass im Hintergrund liegende Flächen einfach überzeichnet wurden. Dies entspricht der, meist als *Painter-Algorithmus* bezeichneten Methode. Dazu müssen die Objekte der Tiefe nach sortiert werden. Allerdings sind Fälle bekannt, in denen diese Technik versagt. Immer dann, wenn die Tiefensortierung der Objekte scheitert, also z.B. bei *zyklischer Überlappung* oder *gegenseitiger Durchdringung*, führt diese Technik zu Darstellungsfehlern, die vom Betrachter als störend empfunden werden. Verschiedene Kundenreklamationen bestätigen die Praxisrelevanz des Problems.

2.3 Anforderungen

Sehr gute Programmierkenntnisse in der Sprache *C*, sowie dreidimensionales Vorstellungsvermögen und Kenntnisse der *Linearen Algebra* waren Grundvoraussetzung für die erfolgreiche Bearbeitung der Aufgabenstellung. Eine zügige Einfindung in das große Softwareprojekt Medina, hinsichtlich Handhabung des Programms, sowie der Quelltexte waren außerdem unverzichtbar zum Lösen des Problems, da nur ein sehr begrenzter Zeitraum zur Verfügung stand. Da die Projektmitarbeiter in der Abteilung sehr unter Kosten- und Zeitdruck stehen, war es für mich außerordentlich wichtig mir auf selbstständiger Basis Wissen und Methoden anzueignen. Glücklicherweise waren trotz Geschäftigkeit immer alle sehr freundlich und hilfsbereit, so dass ich bei unüberwindbaren Problemen, wie den unkonventionellen und leider undokumentierten *Compile-* und *Linkverfahren* stets zufriedenstellende Erklärungen und Anregungen bekam.

3 Der Verlauf des Praktikums

3.1 Die Quellcodeverwaltung

Norbert Frisch, ein externer Mitarbeiter mit dem ich das Büro teilte, hat mich besonders in der Anfangsphase sehr bei der Einfindung in das Projekt Medina unterstützt. Sehr kompliziert und in den ersten Tagen undurchschaubar war vor allem die *Quellcodeverwaltung*, da es unzählige verschiedene Plattformen gibt, auf denen Medina läuft und dann noch eine ganze Reihe von Versionen instand gehalten werden. Hinzu kommen drei verschiedene Entwicklungsumgebungen in denen man sich befinden kann: *prod, dvlp* und *my* genannt. Die erste bezeichnet die Produktionsversion. Hier werden nur Programmteile zurück gestellt, die ausreichend getestet wurden, überall ohne Fehler kompilieren und in die endgültige Version von Medina einfließen. Die dvlp-Umgebung bietet Platz für Entwicklungen, die noch nicht ganz ausgereift sind und auch Fehler enthalten können. Allerdings sollte alles was man dort zurück stellt fehlerfrei kompilieren, da es, wie der prod-Quelltext zentral gespeichert ist und die anderen Entwickler darauf angewiesen sind. So kann es vorkommen, dass man Code schreibt, der auf der eigenen Maschine problemlos übersetzt wird, während ein andere Compiler sich beschwert. Dies kann dazu führen, das man eine entsprechende Rückmeldung von anderen Entwicklern bekommt, die Medina dann nicht mehr kompilieren können. Die my-Umgebung schließlich ist ein Bereich der nur im eigenen *Homeverzeichnis* vorhanden ist. Hier hat man Platz für seine eigene Entwicklung. Dort hält man dann die Quelltexte der Bibliotheken vor, an denen man momentan arbeitet. Erst wenn man weitgehend fertig ist oder andere Entwickler auf die eigenen Ergebnisse angewiesen sind, integriert man die geschriebenen Quelltexte aus der my-Umgebung in den dvlp-Bereich. Die komplette Verwaltung inklusive Befehle zum aus- oder einchecken von Quelltext ist in eigens dafür geschriebenen *Shellskripten* verwirklicht. Dieses Verfahren reicht viele Jahre zurück. Persönlich sehe ich diese Art der Quelltextverwaltung etwas skeptisch, da es besonders für Außenstehende sehr umständlich und gewöhnungsbedürftig ist. Moderne Mittel der Organisation, wie z.B. *CVS* bieten eine Fülle weiterer Features und besitzen zudem noch eine ausgeklügelte und intelligente Versionskontrolle. Doch eine Umstellung bedarf wohl zuviel Aufwand und ist bei dem ohnehin schon knappen Zeitplan der Mitarbeiter nicht zu realisieren.

3.2 Die Projektphasen

Nachfolgend seien die einzelnen Arbeitsphasen des Projektes aufgelistet, auf die in den folgenden Unterabschnitten detailliert eingegangen wird:

Woche	Arbeitsschritt
1	Einführung
1 – 3	Konzeption
1 – 13	Dokumentation
3 – 8	Implementierung
8 – 13	Bugfixing
8 – 13	Optimierung
10 – 13	Tests

3.2.1 Einführung

Nach anfänglichen Schwierigkeiten die Abteilung wiederzufinden (einmal war ich bereits zur Vorstellung dagewesen, allerdings sehen sich die unzähligen Flure im Gebäude zum Verwechseln ähnlich) habe ich mich mit meinem organisatorischen Betreuer Jens Niemann getroffen, der mich erstmal ausführlich in dem Gebäude herumführte. Nachdem er mir eine Einführung in die Arbeits- und Entwicklungsweise des Teams gab, half er mir noch meinen Arbeitsplatz ausfindig zu machen und meine Benutzeraccounts zu beantragen. Die Rechner konnte ich etwa um die Mittagszeit benutzen, da das Anlegen der entsprechenden Konten einen gewissen administratorischen Aufwand darstellte. Den Mitarbeiterausweis hatte ich schon im Vorfeld zugeschickt bekommen. Außerdem habe ich einige Artikel zum Thema Hidden Surface Removal gelesen und mich in das Gebiet der Computergrafik eingearbeitet.

3.2.2 Konzeption

Es wurden verschiedene Vorgehensweisen analysiert und bereits vorhandene Software auf Tauglichkeit hin überprüft. Dabei wurde die Software *GL2PS*[5] eingehend untersucht. Die Idee war an der *OpenGL-Schnittstelle* von Medina anzusetzen und eine bereits vorhandene Software zu verwenden, also „das Rad nicht neu erfinden“ zu müssen. So könnte die Software direkt aus den OpenGL-Instruktionen ein PS-Dokument generieren. Damit ich aussagekräftige Einschätzungen vornehmen konnte, musste ich die Software erstmal in Medina integrieren, was schon einen erheblichen Zeitaufwand darstellte, zumal ich mich zu diesem Zeitpunkt noch kaum im Medina Quellcode auskannte. Leider hat sich herausgestellt, dass sie nicht alle von uns gestellten Anforderungen erfüllt. Beispielsweise waren starke Mängel beim Hidden Surface Removal erkennbar, außerdem zeigten sich Defizite bei der Darstellung von Text. Ein weiterer rechtlicher Punkt ist, dass die Software unter der GPL[6] steht, die sich nicht mit kommerziellen, closed-source Produkten wie Medina verträgt.

Daraufhin war klar, dass ich selber einen Algorithmus schreiben musste, der das Sichtbarkeitsproblem löst. Es waren im Team schon rudimentäre Vorstellungen eines möglichen Algorithmus vorhanden, die als Diskussionsgrundlage dienten. Ab dem dritten Tag fing ich an einen eigenen Algorithmus zu planen.

[5] siehe auch `http://www.geuz.org/gl2ps/`
[6] GNU General Public License

3.2.3 Dokumentation

In der zweiten Woche habe ich eine erste Version der Projektspezifikation erstellt. Diese Spezifikation ist dann in den darauf folgenden Wochen stetig gewachsen und aktualisiert worden. Schließlich habe ich die letzte Woche noch genutzt, um sie zu einer umfangreichen Dokumentation auszubauen. Inbegriffen sind u.a. eine *Ist- und Soll-Analyse*, eine Problembeschreibung, sowie eine Beschreibung des Algorithmus und der Datenstruktur. Ich hoffe, dass die enthaltenden Information zu einem schnellen Verständnis des Algorithmus führen. Die Projektspezifikation ist beigefügt.

3.2.4 Implementierung

Die erste Implemetierungsarbeit hing damit zusammen, die bestehenden Strukturen zu verstehen, um meinen eigenen Code in das Bestehende einfügen zu können. Zuerst habe ich mich damit beschäftigt, die ankommenden Grafikprimitive abzuspeichern und anschließend korrekt auszugeben. Bisher war es so, dass die Grafik direkt ausgegeben wurde. Doch mein Algorithmus verlangt eine Zwischenspeicherung, da die Daten miteinander verrechnet werden.

Ich habe etliche Hilfsfunktionen geschrieben, wie z.B. Schnittpunkt zweier Linien, Kreuzprodukt usw. Diese Funktionen sollten später nützlich sein. Obwohl ich in dieser Phase schon eine Vielzahl der benötigten Funktionen realisiert habe, musste ich später oft bei sehr speziellen Anforderungen neue Hilfsfunktionen schreiben.

Der eigentliche Algorithmus besteht aus einem dreidimensionalen Schneiden, also z.B. wenn zwei Objekte sich gegenseitig durchdringen oder auch bei *zyklischen Überlappungen*. Außerdem werden alle Objekte auch noch zweidimensional miteinander geschnitten. Beispielsweise, wenn zwei Dreiecke sich gegenseitig überlagern, dann muss das hintere entsprechend zurecht geschnitten werden, so dass sich in keinem Punkt der Darstellung mehr als eine Fläche befindet.

3.2.5 Bugfixing, Optimierung und Tests

Die größte Schwierigkeit an dem Projekt war es, unzählige verschiedene Situationen mit möglichst einfachen, performanten Algorithmen richtig zu lösen. Bei der Situation zweier sich gegenseitig überlagernder Dreiecke gibt es eine schier unglaubliche Vielfalt an verschiedenen Stellungen, bei denen das hintere Dreieck mal zu drei Dreiecken, mal zu einem Dreieck wird, oder auch mal ganz verschwindet. Bei den anderen Anforderungen verhält es sich nicht anders. Daher war eine umfangreiche Unteraufgabe unzählige Testmodelle zu entwickeln, die möglichst jede vorstellbare Situation überprüfen. Bei jeder Änderung, war zu befürchten, dass einer der Testfälle nicht mehr funktioniert.

Die Optimiertungsarbeit war ein zentrales Thema, da bei der extrem hohen Anzahl von Vergleichen eine starke Prozessorlast entsteht. Mit *Profilern* ist es möglich herauszufinden in welchen Programmteilen sehr viel Prozessorzeit verwendet wird. Diese Funktionen zu optimieren war sehr wichtig. Was den Algorithmus insgesamt stark verlangsamt hat, war die quadratische Laufzeit. Da jedes einzelne Dreieck potentiell mit

allen anderen Dreiecken verglichen werden muss, entsteht dieses ungünstige Laufzeitverhalten. Aber auch hier waren einige Verbesserungen möglich. So müssen beispielsweise Dreiecke, die beim Schnitt anderer Dreiecke enstanden sind, nicht noch einmal gegen diese getestet werden.

3.3 Unterkunft in Stuttgart

Die drei Monate meines Aufenthalts in Stuttgart habe ich die meiste Zeit in einer größenteils studentischen Wohngemeinschaft in Hohenheim verbracht. Nur den ersten Monat habe ich in einem Gästezimmer bei Verwandten in Sindelfingen übernachtet, von wo aus ich etwa 50 Minuten brauchte, um mit dem öffentlichen Nahverkehr zur Arbeitsstelle zu gelangen. Später waren es bloß noch 30 Minuten. Die schnelle Integration in das studentische Leben hat den Aufenthalt in der fremden Stadt zu einem ausgesprochenen Vergnügen gemacht, welches die Besichtigung der einschlägigen Sehenswürdigkeiten, sowie die ein oder andere kulturelle Erfahrung nicht vermissen ließ. Leider hatte ich neben dieser Vielzahl von erfreulichen Ereignissen auch noch die Unanehmlichkeit drei Prüfungen in Kassel wahrzunehmen, was zu aufwendigen und kostenintensiven Fahrten führte.

3.4 Organisatorisches

Als Mitarbeiter erhält man bei T-Systems einen eigenen Mitarbeiterausweis mit Lichtbild, der zur Identifikation am Eingang, zum Bezahlen im Betriebsrestaurant, sowie der Cafeteria, Benutzung des Parkhauses und Erfassung der Arbeitszeit dient. Außerdem habe ich noch einen eigenen Schlüssel zum Büro erhalten. Dieses Büro war Fremdarbeitskräften vorbehalten. Drei Arbeitsplätze waren vorhanden, von denen einer fast durchgehend von einem anderen externen Mitarbeiter beansprucht wurde. Der dritte Arbeitsplatz wurde nur sporadisch von externen Mitarbeitern benutzt.

Die Arbeitszeiten werden bei T-Systems sehr flexibel gehalten. Es gibt Zeituhren, die die Ankunft und das Verlassen an der Arbeitsstelle erfassen. Das *Gleitzeitverfahren* bietet die Möglichkeit die Arbeitszeit individuell zu gestalten und Überstunden „abzufeiern“. Allerdings werden Zeiten vor 6:00 Uhr und nach 19:00 Uhr nicht erfasst. Eine Kernarbeitszeit ist nicht vorhanden. Bei der 40-Stunden-Woche wurden pro Tag 40 Minuten Pause pauschal von der registrierten Arbeitszeit abgezogen.

Gegessen wird bei T-Systems in der von der Firma *Eurest* betriebenen Kantine. Dort stehen mittags eine Vielzahl von Gerichten zur Auswahl bereit. Obwohl das Essen weitgehend hervorragend war, ist zu bemängeln, dass die Speisen zu meiner gewohnten Essenszeit um 12:30 Uhr oftmals schon vergriffen waren. Da die Kantine um 11:00 Uhr schon öffnet und die Mehrzahl der Mitarbeiter um diese Zeit zu essen pflegen, scheint ein ausreichendes Nachfüllen nach 12:00 Uhr, der Meinung des Betreibers nach, nicht angebracht. Daher haben wir es uns nicht nehmen lassen ab und zu bei einer benachbarten Konkurrenzkantine vorbei zuschauen.

4 Das Ergebnis

4.1 Die Algorithmusidee

Da keiner der bewährten Algorithmen die Anforderungen erfüllte, musste ein neuer Algorithmus entwickelt werden. Dieser kann und sollte sich die bereits erprobten Ideen und Methoden anderer bestehender Verfahren zunutze machen. Abbildung 3 zeigt den schematischen, vereinfachten Ablauf des implementierten Algorithmus. Alle ankommenden TIG-Elemente sollen verarbeitet werden und später in das *CGM-Format* ausgegeben werden. Jedes ankommende Element durchläuft die einzelnen Schritte, dann erst wird das nächste Element betrachtet. Die Objekte können in drei verschiedene Klassen eingeteilt werden:

- *Polygone*
- *Polylines*
- *Hardware-Text*[7]

4.2 Der Ablauf

In den folgenden Abschnitten werden die einzelnen Schritte in der Reihenfolge aus dem Diagramm (Abbildung 3) näher beschrieben. Die *Textelemente* – ausgenommen Softwaretext – erfahren hier eine spezielle Behandlung. Sie werden separat nach den geometrischen Objekten gezeichnet, da sie immer im Vordergrund liegen und sichtbar sein sollten. Alle anderen geometrischen Elemente werden in der Abarbeitung mit den vorhandenen, bereits in die Datenstruktur eingefügten Elemente verglichen und ggf. zugeschnitten oder im Falle eines vollständig überdeckten Elements auch gänzlich gelöscht.

4.3 3D-Schnitt

Der erste Test besteht aus einer Betrachtung der Tiefendimension. Liegen zwei Objekte A und B in z-Richtung disjunkt voneinander, so durchdringen sie sich nicht und es kann zweifelsfrei festgestellt werden, welches der beiden Objekte das andere in der Ansicht überdeckt. Im anderen Fall ist die Situation nicht so klar. Hier wurde durch einen *Min-Max-Test* festgestellt, dass zwei Objekte sich evtl. durchdringen. Außerdem ist ohne weitere Untersuchung nicht eindeutig, welches der beiden Objekte vor dem anderen liegt. Dies ist aber für die Erfüllung der Anforderungen zwingend notwendig.

Ist durch den beschriebenen Test festgestellt worden, dass sich zwei Objekte potentiell durchdringen, muss eine eingehende Untersuchung durchgeführt werden. Schneidet ein

[7] Unter Hardware-Text versteht man bei Medina Objekte, die als Eigenschaften u.A. einen Anker im Koordinatensystem, sowie einen Textstring (ASCII) besitzen. Es ist Aufgabe des darstellenden Geräts den Text an die richtige Stelle zu zeichnen. Software-Text im Gegensatz bezeichnet Schrift, die in Form von Polygonen und anderen Grafikelementen umgesetzt ist.

Dreieck das andere Dreieck im dreidimensionalen Raum, so ist von einer gegenseitigen Durchdringung zu sprechen.

Liegt keine gegenseitige Durchdringung der Objekte vor, so muss doch festgestellt werden, welche der beiden Objekte vor dem anderen liegt. Eines der beiden Objekte ist potentiell teilsichtbar. Teilsichtbare Flächen können bei der späteren Darstellung im PostScript-Format zu Darstellungsfehlern führen. Daher ist es sinnvoll, sich in diesem Schritt zu merken welches der beiden Dreiecke vor dem anderen liegt. Dies wird erreicht indem die Ebenen der beiden Dreiecke untersucht werden.

4.4 2D-Schnitt

Idealerweise sollten Flächen und Linien, die nur zu einem Teil sichtbar sind, entsprechend zugeschnitten werden. Liegt eine Fläche bzw. eine Linie verdeckt unter einer anderen Fläche, so soll der verdeckte Teil entfernt werden, während der sichtbare Teil vorhanden bleibt. Dies erfordert das weitere Unterteilen von Flächen. Also muss der Umriss der oberen Fläche aus der verdeckten Fläche ausgeschnitten werden. Zuerst jedoch muss erkannt werden, welches Dreieck welches andere in der x-y-Ebene überdeckt. Hat man ein sich potentiell überdeckendes Paar erkannt *(Bounding-Box-Test)*, muss ein genauerer Test erfolgen, der die Frage der Überdeckung eindeutig beantwortet. Sind zwei sich gegenseitig überdeckende Dreiecke gefunden, so wird das hintere Dreieck nun zerschnitten. Dieser Vorgang kann zweidimensional stattfinden. Dies ist ausreichend, da die Koordinaten ohnehin schon für die Ansicht auf Bildschirmkoordinaten transformiert wurden.

4.5 CGM-Ausgabe

Nachdem ein Objekt die Verarbeitungsstufen durchlaufen hat, wird es zwischengespeichert. An dieser Stelle befindet sich die Darstellung immer in einem konsistenten Zustand. Das bedeutet insbesondere, dass jede Fläche nur einschichtig sein darf. Kommt nun ein weiteres Dreieck dazu, so werden dahinter liegende Flächen und Linien gelöscht bzw. abgeschnitten und Durchstoßungen aufgelöst, so dass wieder nur einschichtige Flächen bleiben. So wird das Bild sukzessive aufgebaut bis schließlich keine weiteren Elemente dazukommen.

Schließlich erfolgt die Ausgabe in folgender Reihenfolge:

1. *Dreiecke*
2. *Linien*
3. *Hardware-Text*

Innerhalb dieser Gruppen ist es nun egal in welcher Reihenfolge sie ausgegeben werden. Da keine sich überdeckenden Flächen existieren, können sie sich auch nicht gegenseitig übermalen. Die Linien werden über die Dreiecke gezeichnet. So wird sichergestellt, dass die Linien nicht von den Rändern der Dreiecke überdeckt werden können. Die zuvor

gesammelten Textobjekte werden zum Schluss ausgegeben. Sie sollen in jedem Fall ganz zuletzt dargestellt werden, da sie immer im Vordergrund und damit sichtbar sind.

In den Abbildungen 4 und 5 soll die Wirkungsweise des Algorithmus dargestellt werden. In der ersten Grafik sieht man einen mit dem *HSR-Algorithmus* erstellten Plot. Abbildung 5 ist leicht gedreht. Gut zu erkennen ist das Fehlen der im Hintergrund befindlichen Teile des Modells, ähnlich dem scharfkantigen Schattenwurf einer Lichtquelle.

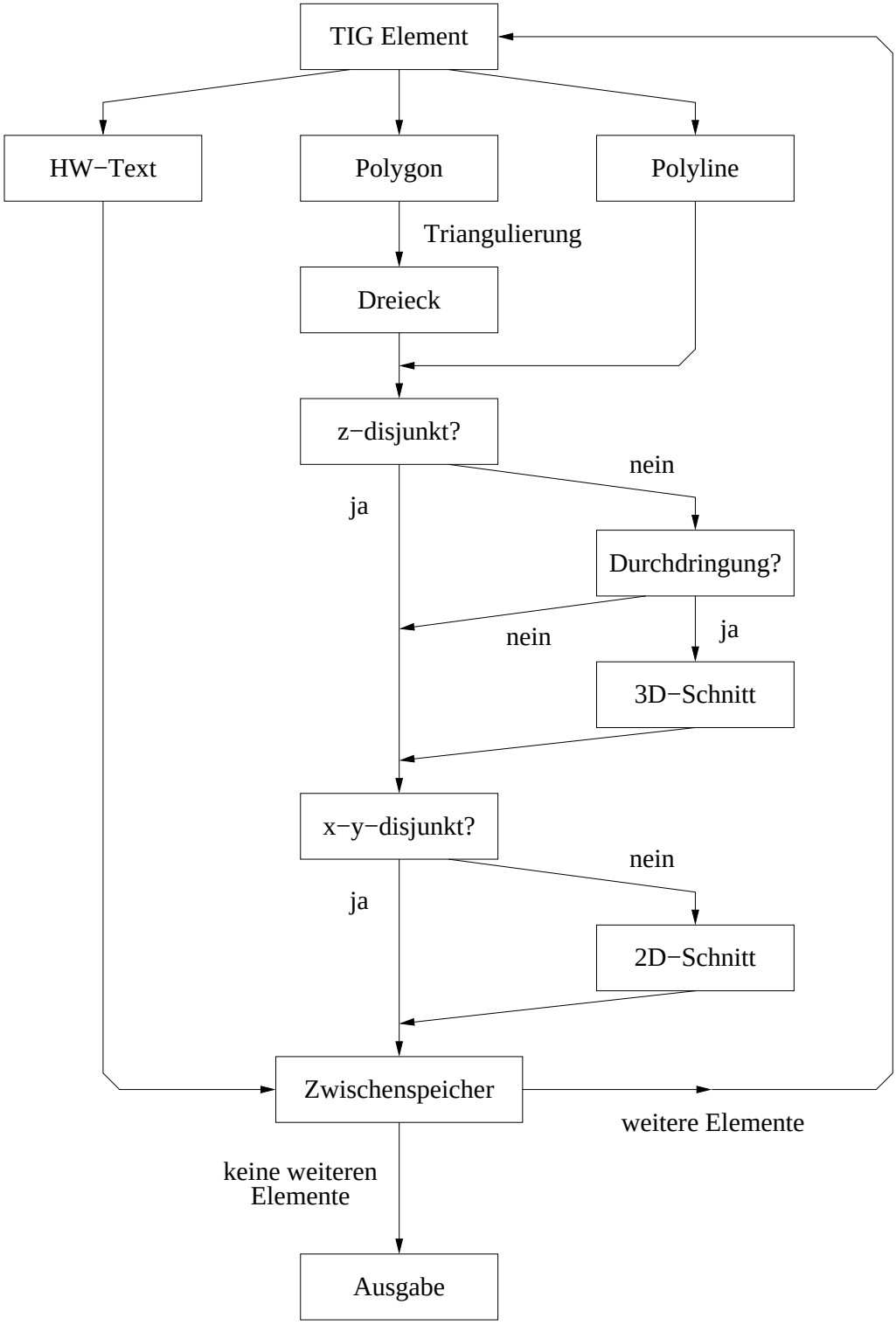

Abbildung 3: Ansicht mit *HSR Algorithmus*

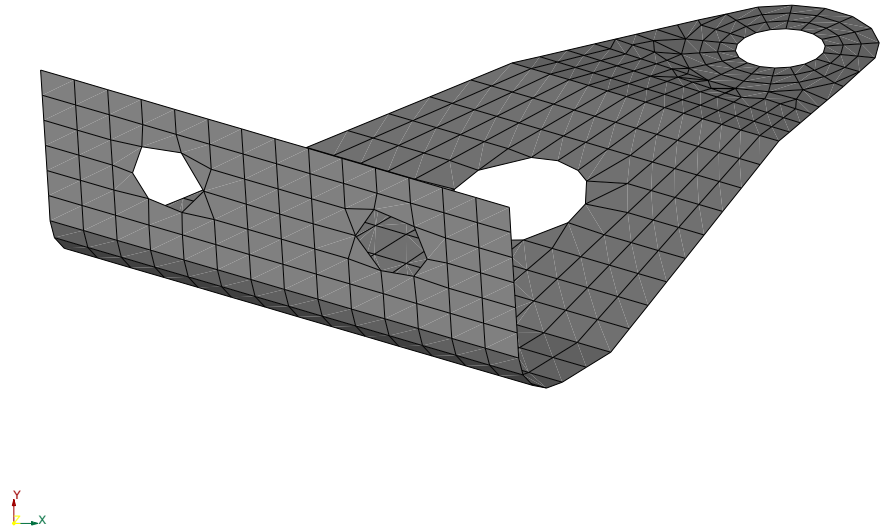

Abbildung 4: Ansicht mit *HSR Algorithmus*

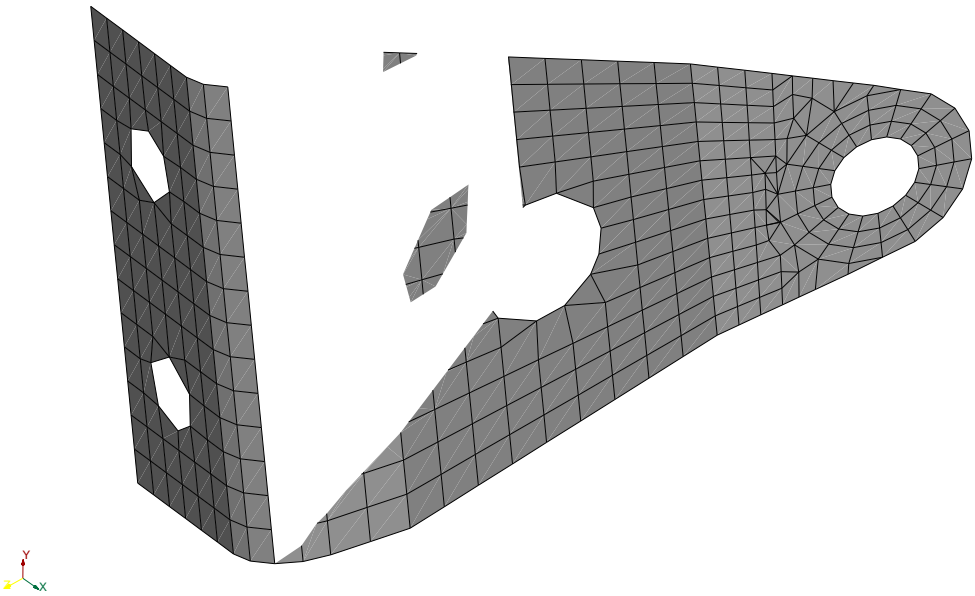

Abbildung 5: Ansicht mit *HSR Algorithmus* (gedrehte Perspektive)