Praktikumsbericht
1. Praxissemester

Norman Walter
Matrikelnummer: 728297

FHTE Esslingen
Studiengang
Softwaretechnik/Medieninformatik

September 2005 - Februar 2006

Durchgeführt bei

GmbH Backnang
Gerberstraße 33
71522 Backnang

Abteilung: Wireless Product Unit (WPU/HWA4)

Teil 1: gds nach Ie3d Konverter

Betreuer: Dr. Stefan Kern

Inhaltsverzeichnis

Einleitung	4
Ist Analyse	5
Bedienungsanleitung	7
Use case Diagramm	10
Use Case Diagramm – Ergänzungen	11
Probleme der aktuellen Programmversion	12
Deckungsgleiche Polygone	13
Die Suche nach dem Rundungsfehler	18
Falsche physikalische Parameter in den dielektrischen Layern	22
Layer mehrfach verwenden	24
Verbesserung des Laufzeitverhaltens	27
Implementierung des neuen IE3D XML 2.0 Formats	29
Backus Naur Form des GDS Formats	32
Portierung auf gcc (g++)	33
Die neue Timer Klasse	34
Makefile für gds2ie3d	35
Zeitplan	37
Meilensteine	37
Literaturverzeichnis	39

Einleitung

Mein 1. Praxissemester absolvierte ich bei der Firma Ericsson (ehemals Marconi) in Backnang.
An diesem Standort werden unter anderem Richtfunkgeräte zur Datenübertragung entwickelt.
Das Praktikum wurde in der Abteilung "Wireless Product Unit / Stratetic Concepts" durchgeführt.
Diese Abteilung beschäftigt sich mit der Entwicklung von Hochfrequenzschaltungen für den Richtfunk.

Die Schaltungen werden am Computer mit Hilfe des CAD *(Computer Aided Design)* Programms CADENCE, welches auf Sun Workstations (Ultra Sparc Architektur) unter dem Betriebssystem Solaris läuft, entworfen und mit einem Feldsimulator *(z.B. IE3D)* auf ihr Verhalten hin überprüft.

Meine Aufgabe war es ein bereits existierendes Konvertierungsprogramm zu pflegen. Dieses Konvertierungsprogramm übernimmt die Aufgabe, die von CADENCE gelieferten zweidimensionalen CAD Daten unter Verwendung einer sogenannten "Technologiedatei" in ein dreidimensionales Modell für den Feldsimulator IE3D zu überführen. Der Konverter verschafft dem Unternehmen einen erheblichen Wettbewerbsvorteil, da Entwürfe, die für die Produktion angefertigt werden, so direkt in den Feldsimulator übernommen werden können, ohne dass jemand ein zweites Modell anfertigen muss. Dadurch wird enorm viel Zeit bei der Entwicklung der Schaltungen eingespart.

Das Konvertierungsprogramm lag dabei in Form von C++ Quellcode vor. Außerdem haben diverse Vorgänger (Diplomanten und Praktikanten) eine Dokumentation dazu angefertigt.

Ist Analyse

Da meine Vorgänger bereits eine sehr ausführliche Dokumentation erstellt haben, möchte ich diesen Abschnitt kurz fassen.

Beim Entwurf der Schaltungen mittels CADENCE werden zweidimensionale Polygone auf verschiedenen Ebenen (genannt *Layers*) abgelegt. CADENCE erlaubt es, den Schaltungsentwurf in einer gds *(Graphics Display System)* Datei zu exportieren.

Zusätzliche Angaben über die dritte Dimension (z Koordinaten), sowie physikalische Parameter liegen in einer sogenannten Technologiedatei vor.

Das Konvertierungsprogramm gds2ie3d erzeugt nun aus den zweidimensionalen Polygonen der gds Datei unter Zurhilfenahme der Parameter aus der Technologiedatei ein dreidimensionales Modell. Dieses Modell wird dann im geo Dateiformat des Feldsimulators IE3D gespeichert.

Der Vorgang, bei dem aus zweidimensionalen Polygonen ein dreidimensionales Modell entsteht, wird im folgenden als *Aufwachsen lassen* bezeichnet.

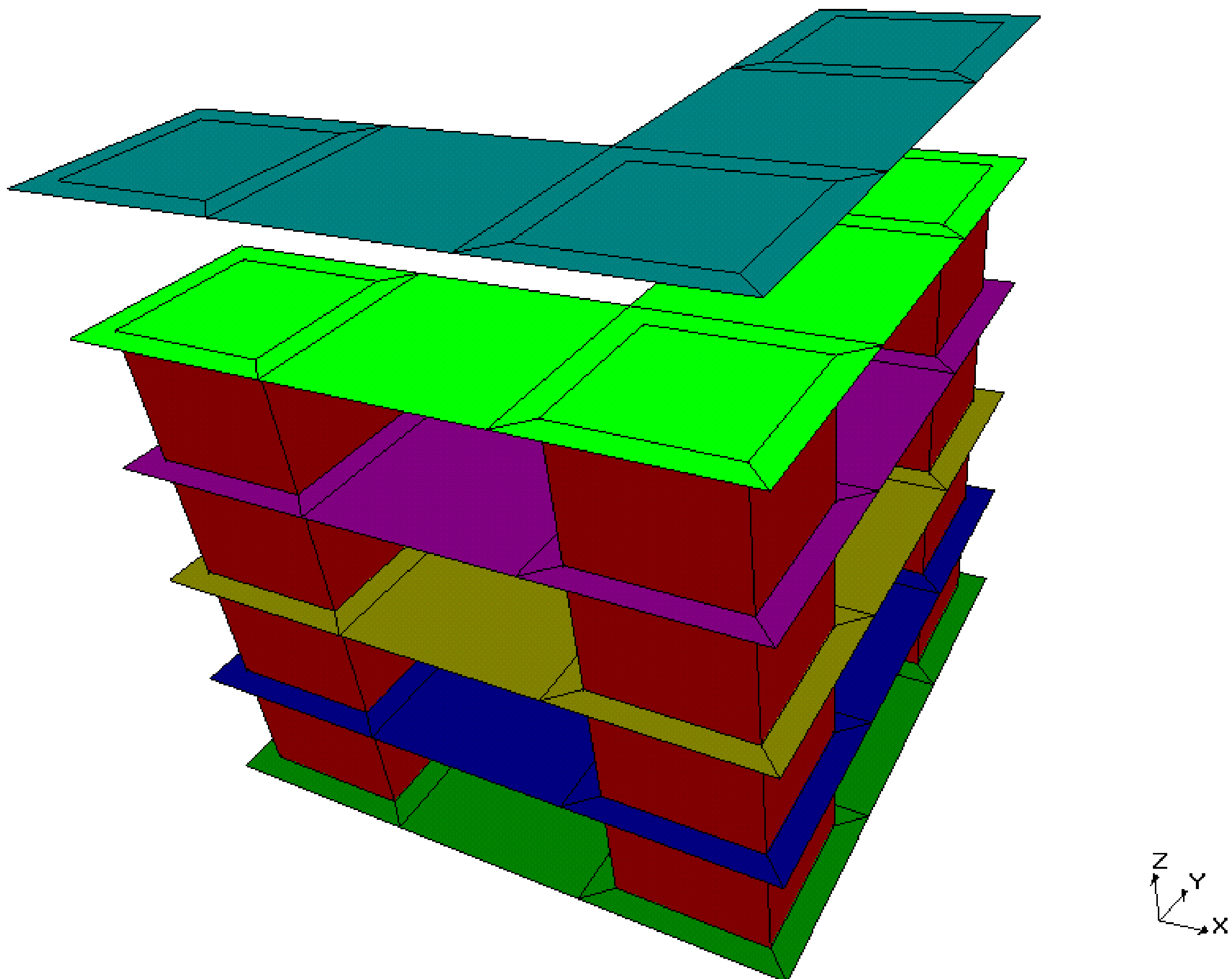


Abbildung 1: Mgrid 3D Darstellung von IE3D

Abbildung 1 zeigt die 3D Darstellung von IE3D. Die gezeigte Schaltung wurde als Testfall für das Konverterprogramm entworfen. Es handelt sich dabei um eine einfache *Multilayer* Schaltung. Sie besteht aus 5 Schichten Isolatormaterials und metallischen Leiterbahnen. Die isolierenden Schichten zwischen den Leiterbahnebenen werden nicht dargestellt. Die einzelnen Ebenen sind elektrisch über sogenannte VIAs verbunden. Diese VIAs sind in dem Beispiel als Quader mit quadratischer Grundfläche dargestellt.

Der Quellcode des Konverters besteht aus über 300000 Zeilen Programmcode, der in verschiedene Module aufgeteilt ist. Dabei steht klar das objektorientierte Design im Vordergrund: Alles ist fein säuberlich in Klassen unterteilt und somit gekapselt. Von den C++ Sprachmitteln wie Klassen, Methoden, Vererbung, Abstrakte Datentypen usw. wird reichlich Gebrauch gemacht. Auch die *Standard Template Library* (STL) kommt häufig zum Einsatz.

Section RecordHeader BaseObject Polygon3Dtable SectionContainer
Polygon3D GDSifstream GDSRecord EdgeContainer ElementContainer GDSStreamFormat IE3DKonverter PolygonContainer
StructureContainer StructureElement
Polygongen
GDSXY GDSWidth
GDSLayer GDSHeader GDSEndstr GDSPolygon GDSBgnlib GDSBgnstr GDSBoundary GDSDatatype GDSEndel GDSEndlib
GDSPathtype GDSLibname GDSPath GDSPolygon GDSReflibs GDSStrname
BoundaryElement PathElement

GDSDate BaseError Complex MessageManager GEORecords Option Coord2D Coord3D Parser Edge
EdgeGEO
GEOGridandWalls GEOExternPorts GEODiellayer GEOGlobal GEOMetaltype

Abbildung 2: Klassensystem des Konverters

Bedienungsanleitung

Beschreibung der Parameter

gds2ie3d.exe oder gds2ie.exe

Erforderliche Parameter

- s Dateiname

Gibt die GDS Datei an. Dateiname muss ein vollständigen Pfad und Dateiname+Erweiterung der GDS Datei enthalten.

- tf Dateiname

Gibt die Technologiedatei an. Dateiname muss ein vollständigen Pfad und Dateinamen+Erweiterung enthalten.

oder :

- t {triquint | ums | alk}

Wird –t gefolgt von einem der zwischen den geschweiften Klammern angegebenen Parameterwerten übergeben, so wird damit (zusammen mit dem Parameter –g) die zu verwendende Standardtechnologiedatei definiert.
Dieser Parameter erfordert das Vorhandensein der entsprechenden Standard tek Datei (in Abhängigkeit mit dem Parameter –g) im Verzeichnis m:\konverter\tek

Optionale Parameter:

-g

Wird –g angegeben, so generiert die Anwendung aus der flachen Struktur eine 3D-Struktur, sofern in der Technologiedatei die Option grow für die beteiligten Polygone nicht auf Null gesetzt ist.

-a Dateiname

Gibt die GEO Datei an. Dateiname muss ein vollständiger Pfad und Dateiname+Erweiterung enthalten.

- i Dateiname

Gibt die INI Datei an. Dateiname muss ein vollständigen Pfad und Dateinamen+Erweiterung enthalten.

- tcl

Wird dieser Parameter angegeben, so werden alle Ausgaben in eine Datei, welche in der INI Datei angegeben ist, umgeleitet, anstatt sie auf dem Bildschirm auszugeben.

Verzeichnisse

m:\konverter	\cfg\technologie.cfg	<- Konfigurationsdatei für den Parser
	\bin\gds2ie3d.exe	<- Anwendung
	\bin\default.ini	<- Standart INI Datei
	\tek*.tek	<- Standart tek Dateien

INI Datei

Die INI Datei dient zum spezifizieren der Meldungsausgaben des Konverters. Alle wichtigen Ausgaben und Statusmeldungen werden auf dem Bildschirm ausgegeben. Zusätzlich kann das Programm Meldungen und Hinweise auch in mehrere Dateien schreiben.[vgl. Parameter –tcl]
Diese Dateien werden Log Dateien genannt.

Die wichtigste Datei ist die Datei Konverter Log Datei. In sie werden alle Statusmeldungen und Fehler geschrieben.
Die restlichen Log Dateien sind zum überprüfen des internen Programmablaufes.
Der Pfad für das Konverter LOG File muss zwingend existieren! Bei allen anderen Pfade kann eine leere Zeile angegeben werden, was bedeutet, das die Log Ausgabe dann unterdrückt wird.

Die INI Datei sollte wie folgt aufgebaut sein. Leerzeichen am Ende der Zeile sind nicht erlaubt:

```
erste Zeile ist eine Kommentarzeile
Pfad für TEK-LOG Datei
Pfad für GDS-LOG Datei
Pfad für GEO-LOG Datei
Dateiname und Pfad der Konverter LOG
```

Rückgabewerte

Folgende Rückgabewerte werden von gds2ie3d.exe an das Betriebsystem zurückgegeben:

Beschreibung	Rückgabewert
Programm korrekt beendet; keine Fehler	0
Programm mit schwerem Fehler abgebrochen	1 / -1
Programm mit einer Warnung beendet	2

Wird das Programm mit einer Warnung beendet, so bedeutet das, dass das Programm erfolgreich ausgeführt wurde, das Programm aber einige Dinge nicht so vorgefunden hat, wie erwartet wird. (z.B. INI Datei fehlt; Standartwerte angenommen)

Als Alternative zum Aufruf des Konverters per CLI steht ein grafisches Frontend Namens *„Dataman"* zur Verfügung.

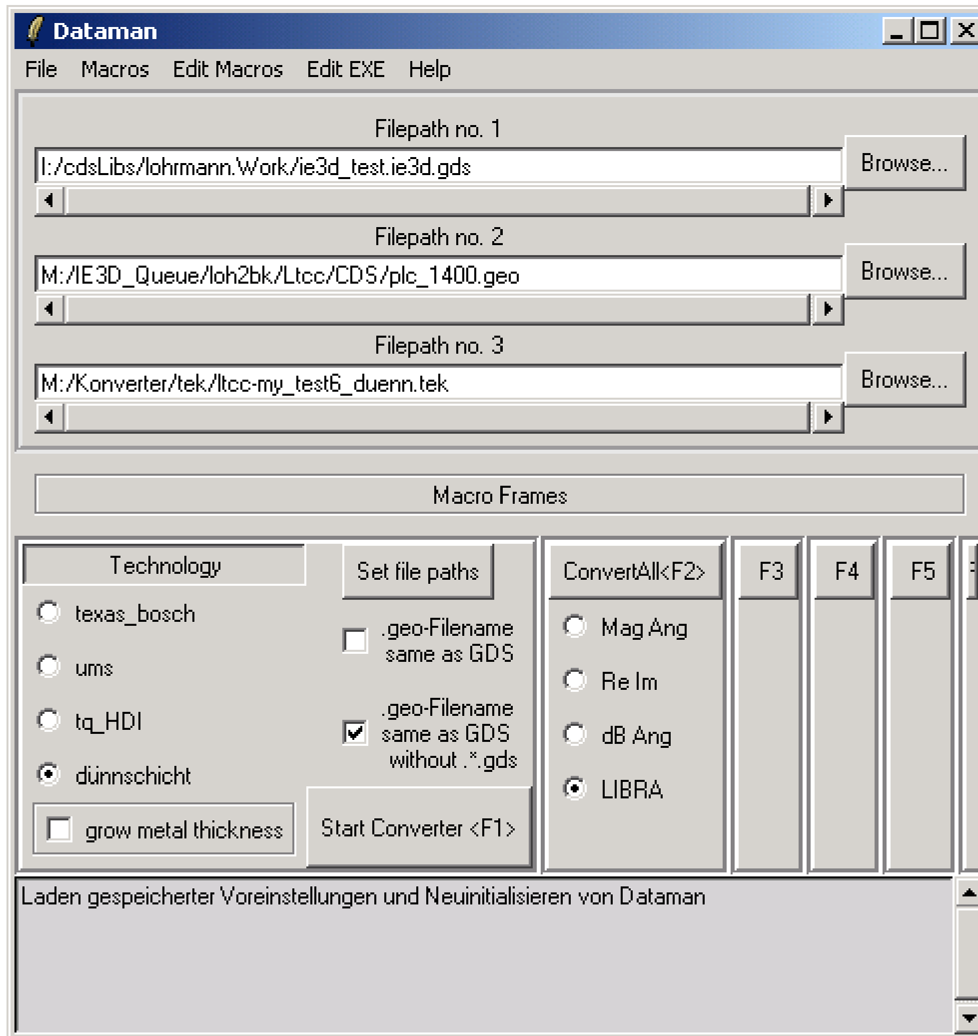
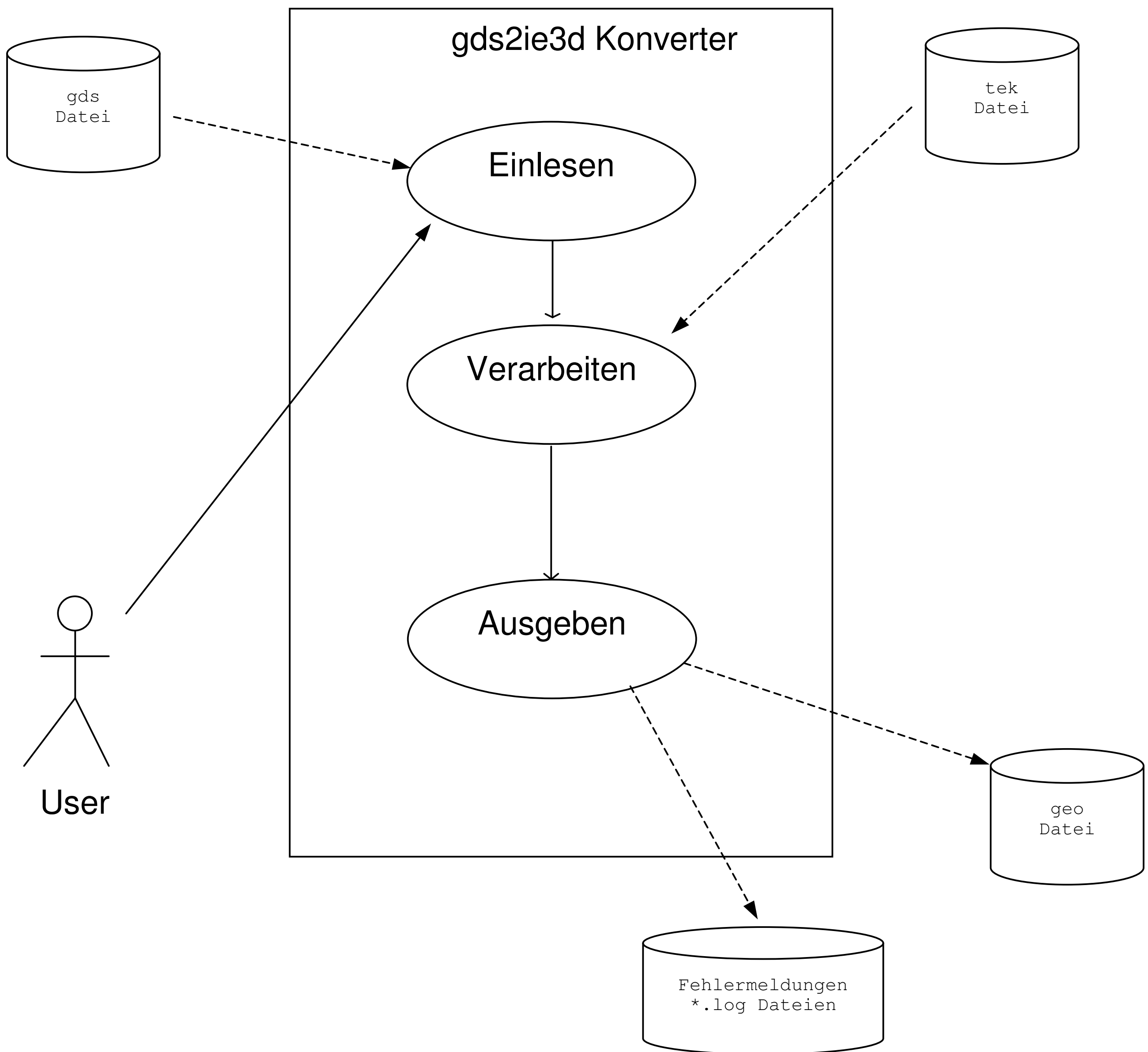


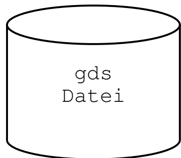
Abbildung 3: Die Grafische Oberfläche „Dataman"

Im unteren Bereich des Fensters werden Hilfsinformationen zu den einzelnen Bedienelementen angezeigt. Nach Beendigung des Konverters wird dort außerdem der Text aus der Datei Konverter.log dargestellt. In dieser Datei befinden sich auch etwaige Fehlermeldungen des Konverters.

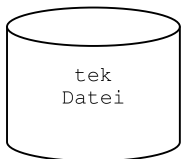
Use case Diagramm



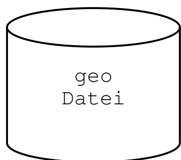
Use Case Diagramm - Ergänzungen



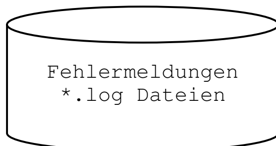
Die gds Datei ist die Eingabedatei. In Ihr stehen die geometrischen Informationen der elektrischen Struktur. Alle Informationen sind in 2D



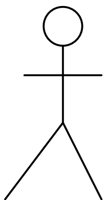
Die tek Datei enthält Informationen für die Verarbeitung der Daten. In dieser Datei steht, welche Eigenschaften die Layer haben, wie die 2D Daten der gds Datei zu interpretieren sind (Externe Ports, 3D Objekte)



Die geo Datei ist die Ausgabe. Diese Datei wird vom Konverter Programm erzuegt. Sie ist direkt im IE3D Simulator zu lesen.



Dies sind mehrere Dateien. Die Ausgabe von Statusmeldungen erfolgt in eine log Datei. Auch werden Fehler in der log Datei protokolliert.



Der User. Das Programm muss von ihm angestoßen werden. Das Programm hat weder Ereignishandling noch Interaktion mit dem Benutzer. Einmal mit den korrekten Daten gestartet arbeitet das Programm völlig selbständig.

Probleme der aktuellen Programmversion

- Unter bestimmten Umständen enthält die von CADENCE gelieferte gds Datei deckungsgleiche Polygone. Diese sollen nach Möglichkeit automatisch herausgefiltert werden.

- Probleme können auch dadurch entstehen, dass sich Polygone überlappen.

- Es treten Abweichungen in den z Koordinaten der geo Datei auf. Ob dies ein Rundungsfehler ist, ist noch nicht ganz klar.

- Die physikalischen Parameter wie Leitfähigkeit usw. werden nicht korrekt aus der Technologiedatei übernommen. Das 3D Modell ist physikalisch gesehen ein massiver Metallklotz, so dass man die entsprechenden Parameter von Hand korrigieren muss.

- Für den Boden und Deckel eines Körpers müssen im derzeitigen System in der Technologiedatei jeweils ein separater Layer eingefügt werden. Es ist nicht möglich dass ein Layer zweimal mit unterschiedlichen z Koordinaten vorkommt. Das Programm soll so verändert werden, dass man ein und denselben Layer mehrfach verwenden kann.

Weitere Aufgaben waren:

- Steigerung der Geschwindigkeit

- Implementierung des neuen IE3D XML 2.0 Formats für die Ausgabe der geo Dateien

Langfristig wird außerdem angestrebt, das Programm portabel zu machen. Die vorliegende Version wurde mit dem Borland C++ Builder 4.0 entwickelt und enthält einige Merkmale, die es von dieser pro Umgebung abhängig machen.

Außerdem enthält das Programm an manchen Stellen Low Level Code (auch x86 Assembler Sequenzen), welcher umgeschrieben werden müsste, damit es auf anderen Architekturen wie z.B. Ultra Sparc oder PowerPC läuft. Dabei ist auch auf die Byte Order zu achten: x86 Architekturen nutzen *Little Endian*, Ultra Sparc oder PowerPC dagegen *Big Endian.*

Deckungsgleiche Polygone

Überladen der Vergleichsoperators der Klasse Polygon3D

Für Grafikprimitiven wie Polygone und Pfade existieren im Programm zwei Klassen Namens Polygon3D und PolygonGEO.

Um deckungsgleiche Polygone zu ermitteln, habe ich den Vergleichsoperator für diese Klassen überladen:

```
// Neu: Überladen des Vergleichsoperators für die Klasse
// Polygon3D. Gleich soll hier deckungsgleich bedeuten.
// Operator ist true, falls Polygone deckungsgleich,
// sonst false.
bool Polygon3D::operator==(const Polygon3D & p) const;
```

Damit gelten zwei Objekte vom Typ Polygon3D als gleich, falls die Polygone deckungsgleich sind.
Die Polygon3D bzw. PolygonGEO Klasse enthält einen STL vector, der als Elemente Objekte der Klasse Edge aufnimmt.

```
vector<Edge>        ingr;        // containes all edges
```

Die Klasse Edge repräsentiert Kanten. Für die Klasse Edge wurde bereits der Vergleichsoperator überladen

```
bool        operator==(const Edge & e) const;
```

so dass man sehr einfach Kanten vergleichen kann.

Damit kann man dann folgende Aussage machen: Zwei Polygone sind genau dann deckungsgleich, wenn all deren Kanten deckungsgleich sind. Sei *V* die Menge der Knoten (Ecken) und *E* die Menge der Kanten eines Graphen (hier eines Polygons).

$$A = (V, E) \quad B = (V, E)$$

Die Graphen A und B sind genau dann deckungsgleich, wenn gilt:

$$\left(\left|E \in A\right| = \left|E \in B\right|\right) \wedge \forall E \in A \exists \left(E \in B = E \in A\right)$$

Das folgende Beispiel zeigt zwei Polygone A und B, die deckungsgleich sein sollen, aus Darstellungsgründen aber nebeneinander gezeichnet sind:

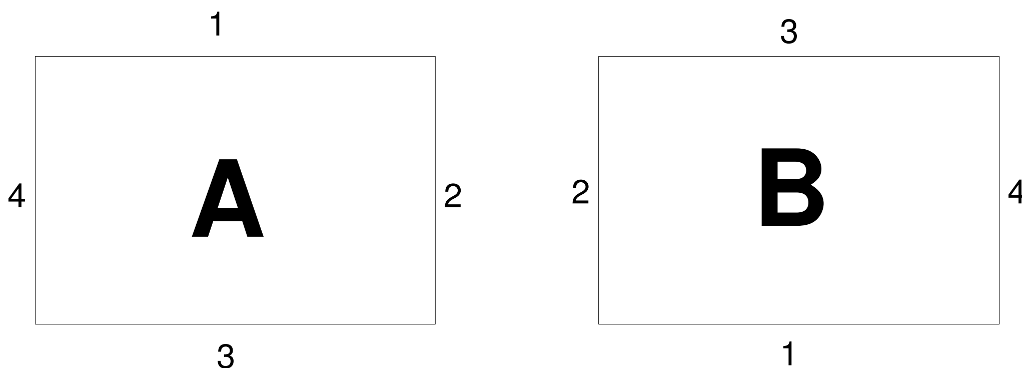


Abbildung 4: Beispiel für zwei deckungsgleiche Polygone

Wenn man die Kanten zweier Polygone A und B vergleicht, muss man folgendes beachten:
Da die Kanten als Edge Objekte in STL Vektoren abgelegt sind, müssen deckungsgleiche Kanten nicht zwingend an der selben Position im Kanten Vektor von Polygon A und B sein. Wie die Skizze oben zeigt, ist bei Polygon B sozusagen die Reihenfolge der Kanten um 180° verdreht. Folglich wäre z.B. Kante 1 von Polygon A nicht deckungsgleich mit Kante 1 von Polygon B, aber Kante 1 von Polygon A wäre hier deckungsgleich mit Kante 3 von Polygon B.

Aus diesem Grund muss man jede Kante von Polygon A mit jeder Kante von Polygon B vergleichen – und zwar so lange, bis man die zur Kante aus Polygon A deckungsgleiche Kante in Polygon B gefunden hat. Falls es zu einer Kante aus Polygon A keine deckungsgleiche Kante in Polygon B gibt, sind die Polygone mit Sicherheit nicht deckungsgleich.

Der Algorithmus lautet demnach wie folgt:

1.) Falls Polygon3D Objekte unterschiedliche Anzahl von Kanten enthalten, gebe false zurück.
2.) Für jede Kante von Polygon A mache folgendes:
3.) Vergleiche diese Kante mit jeder Kante von Polygon B.
4.) Falls diese Kante übereinstimmt setze Rückgabewert auf true und fahre mit Punkt 2 fort. Ansonsten setze Rückgabewert auf false.
5.) Falls man eine Kante von Polygon A mit allen Kanten von Polygon B verglichen hat und keine Übereinstimmung gefunden wurde setze Rückgabewert auf false und breche ab.
6.) Der Rückgabewert enthält nun das Ergebnis der Vergleichsoperation und liegt als boolscher Wert vor (true oder false).

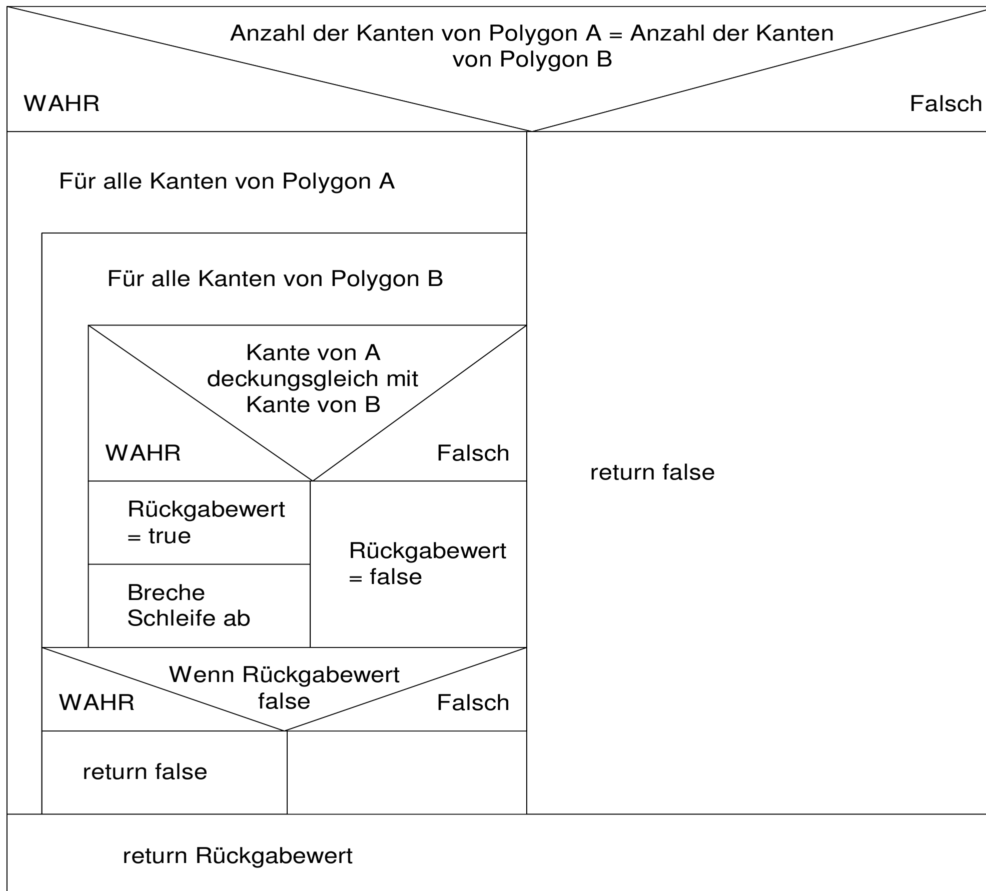


Abbildung 5: Nassi-Schneiderman Diagramm für Polygonvergleich

Bei unserem Beispiel mit zwei Vierecken würden folgende Vergleiche von Kanten stattfinden:

A1 == B1	A1 == B2	A1 == B3	A1 == B4
A2 == B1	A2 == B2	A2 == B3	A2 == B4
A3 == B1	A3 == B2	A3 == B3	A3 == B4
A4 == B1	A4 == B2	A4 == B3	A4 == B4

A1 bezeichnet hier Kante 1 von Polygon A, B2 Kante 2 von Polygon B und so weiter.
Wir brauchen also zum Vergleich zweier Polygone A und B mit n bzw. m Kanten m mal n Vergleichsoperationen. Wenn wir annehmen, dass die zwei Polygone die gleiche Anzahl Kanten besitzen, können wir uns damit leicht die Komplexität des beschriebenen Algorithmus für den worst case herleiten:

$O(n^2)$ *(worst case)*

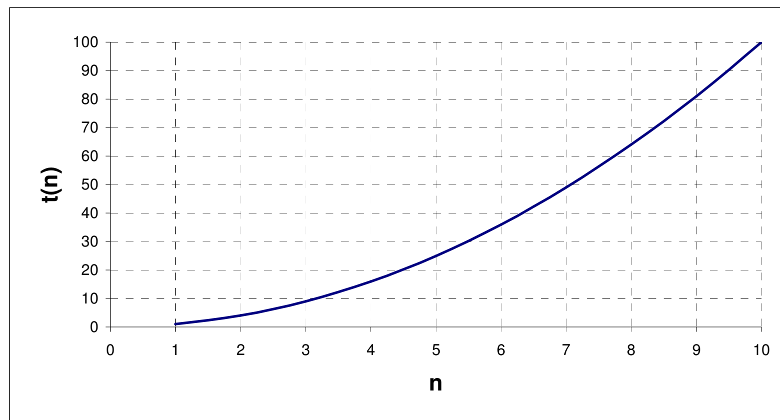


Abbildung 6: Benötigte Rechenzeit für n Kanten

Für den Vergleich zweier Dreiecke werden beispielsweise $3^2 = 9$ Zeiteinheiten benötigt, für zwei Vierecke $4^2 = 16$ und so weiter.

Schreibtischtest des Vergleichsoperators der Klasse Polygon3D

Es folgt ein Schreibtischtest des beschriebenen Algorithmus zum Vergleich zweier Polygone.
Als Eingabedaten werden die zwei Beispielpolygone A und B verwendet.

A1 == B1 false	A1 == B2 false	A1 == B3 true		retval = true
A2 == B1 false	A2 == B2 false	A2 == B3 false	A2 == B4 true	retval = true
A3 == B1 true				retval = true
A4 == B1 false	A4 == B2 true			retval = true

Nach dem letzen Durchgang ist der Rückgabewert retval true, was bedeutet dass die beiden Polygone deckungsgleich sind. Dies entspricht auch der Wahrheit.

Die Methode ContainsPolygon3D der Klasse Polygon3Dtable

In der Klasse Polygon3dtable habe ich folgende Methode eingeführt:

```
bool ContainsPolygon3D(const unsigned int layer, const Polygon3D & p);
```

Damit kann geprüft werden, ob im angegebenen Layer bereits ein zum Polygon p deckungsgleiches Polygon existiert. Die Methode liefert true, falls ein solches deckungsgleiches Polygon existiert, sonst false.

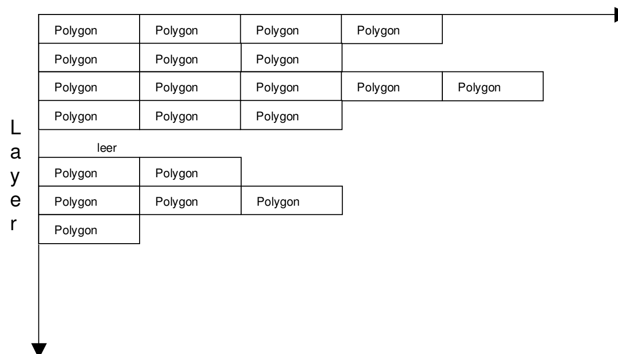


Abbildung 7: Anordnung der Polygon3D Objekte in den Layers

Der verwendete Algorithmus läuft wie folgt ab:

1.) Für jedes Polygon im Layer mache folgendes:
2.) Prüfe, ob Polygon deckungsgleich mit Polygon im Layer ist. Falls ja, gebe true zurück. Ansonsten fahre mit Punkt 1 fort.
3.) Gebe false zurück.

Der Vergleich der Polygone findet mit Hilfe des überladenen Operators == aus der Klasse Polygon3D statt.

Die Komplexität des verwendeten Algorithmus ist linear. Das bedeutet die Laufzeit hängt linear mit der Anzahl n der Polygone im jeweiligen Layer zusammen. Im schlimmsten Fall sind n Vergleiche nötig. Dabei ist allerdings zu beachten, dass die Vergleichsoperation im schlimmsten Fall quadratische Komplexität (abhängig von der Anzahl der Kanten der Polygone) hat.

$O(n)$ *(worst case)*

$O(n^3)$ *(unter Berücksichtigung der Komplexität des Vergleichsoperators)*

Modifikation der Methode push_back der Klasse Polygon3Dtable

In der Klasse Polygon3Dtable gibt es die Methode push_back , die Objekte vom Typ Polygon3D in die Polygon3Dtable einfügt.

```
void push_back(const unsigned int layer,const Polygon3D & p);
```

Diese wurde so verändert, dass nunmehr deckungsgleiche Polygone ignoriert werden sollten.

Dazu wird mit Hilfe der Methode ContainsPolygon3D der Klasse Polygon3Dtable zunächst geprüft, ob für das einzufügende Polygon schon ein deckungsgleiches Polygon im entsprechenden Layer existiert.
Falls ja, wird das Polygon nicht eingefügt.

```
if (ContainsPolygon3D(layer,p))
{
  // Falls deckungsgleiches Polygon im Layer enthalten ist
  cout << "Ignoriere deckungsgleiches Polygon" << endl;
}
else
{
  // Falls nicht...
  // Füge Polygon3D Objekt in Layer ein
```

Diese Vorgehensweise bringt folgenden Seiteneffekt mit sich: Da Instanzen der Klasse Polygon3D nicht nur dreidimensionale Polygone, sondern auch Pfade *(gds Grafikprimitive „Path“)* enthalten können, werden auch deckungsgleiche Pfade aus der gds Datei ignoriert.

Dadurch, dass überflüssige Objekte nicht mehr in die geo Datei aufgenommen werden, wird diese auch deutlich kleiner. Bei der Testdatei "via_wall_overlay_ie3dtest.ie3d.gds" (siehe auch Abbildung 1) konnte die Größe der resultierende geo Datei von 42k auf 38k reduziert werden.

Die Suche nach dem Rundungsfehler

In der mir vorliegenden Version des Konverters war ein Rundungsfehler, der dazu führte, dass bestimmte Werte nicht korrekt in die Ausgabedatei geschrieben wurden. Das führte dazu, dass der Benutzer diese falschen Werte nach der Konvertierung von gds nach geo in IE3D von Hand korrigieren musste.

Zunächst war unklar, an welcher Stelle im Programm dieser Fehler zu suchen ist.
Deshalb habe ich mir sämtliche fehleranfälligen Funktionen, die an der Verarbeitung dieser Werte beteiligt sind angeschaut und teilweise komplett neu geschrieben.

Überarbeitung von globalutilities.cpp

In der Include Datei globalutilities.cpp wurden einige Funktionen neu implementiert.
Zur Vereinfachung wurde eine neue Funktionsschablone Namens convert geschaffen:

```
// Neu: convert konvertiert vom gegebenen Typ in
// den gewünschten Typ. Der Typ wird automatisch aus
// den Datentypen der eingesetzten Variablen ermittelt.
template <class type_given, class type_wanted>
type_wanted convert(type_given &val, type_wanted &typeVar)
{
  stringstream convStream;
  type_wanted var;
  convStream << val << endl; // Wert in den Datenstrom einfügen
  convStream >> var;            // Wert im gewünschten Typ aus dem Datenstrom auslesen
  return var;
}
```

Mit Hilfe dieser Funktionsschablone kann man jeden Datentyp in jeden anderen Datentyp konvertieren.
Der Trick dabei ist, dass die Funktionsparameter abstrakte Datentypen sind.

Folgende Funktionen wurden damit neu geschrieben:

```
string int2string(int I)
{
   string retval;
   return convert(I,retval);
}

int string2int(const string &s)
{
  int retval;
  return convert(s,retval);
}

double string2float(const string & s)
{
  double retval;
  return convert(s,retval);
}

double string2scientific(const string & s)
{
  double retval;
  return convert(s,retval);
}
```

Um den Fehler zu suchen habe ich unter anderen den Wert des Parameters EdgeCell verfolgt, der aus der Technologiedatei gelesen wird und ohne irgendwelche Verrechnungen 1:1 an die geo Ausgabedatei weitergegeben werden sollte. Hier der entsprechende Ausschnitt aus der Technologiedatei „ltcc-koppler_5flat.tek“:

```
########################################
#              LTCC              #
########################################
#                Globale Parameter          #
########################################

[Global]
        formatversion        ==        3.15
        CommentLine          ==        NoComment
        Commentline          ==        nocomment
        AutoEdgeCell         ==        No
        EdgeCell             ==        2.0e-2
        MaxFreq              ==        40
        Cal                  ==        1.5e+3
        Cells                ==        20
        TopEnclosure         ==        No
        BottomEnclosure      ==        No
        LeftEnclosure        ==        no
        RightEnclosure       ==        no
        Plnwave              ==        no
```

Der Wert für den Parameter EdgeCell wird hier in als 2.0e-2 angegeben. In der resultierenden geo Datei erscheint aber folgendes:

```
3.15 0 0 1.9999999553e-02 0
4.0000000000e+01 1 0 2 4 7 18 1.5000000000e+03
```

Die vierte Zahl entspricht dem Wert des Parameters EdgeCell . Wie man sieht, steht hier statt des erwarteten Werts 2.0e-2 der Wert 1.9999999553e-02.

Bei genauerer Betrachtung des überladenen Operators << in geoutility.cpp hat sich herausgestellt, dass hier der Fehler liegt. Der Wert wird korrekt durchgereicht, aber mit Rundungsfehler in den Ausgabestrom geschrieben.

```
// Schreibt die erste Zeile der geo Datei
ostream & operator<<(ostream & os,const global_first_line & f)
{

    long old=os.setf(ios::fixed,ios::floatfield);
    os.precision(2);
    os<<" "<<f.version<<" "<<f.num_of_comments<<" "<<f.auto_edge_cell<<" ";
    os.setf(ios::scientific,ios::floatfield);
    os.precision(10);

    // Test: Stimmt der Wert für r.edge_cell_w noch?
    cout << "geoutility.cpp:" << endl;
    cout << "f.edge_cell_w=" << f.edge_cell_w << endl;

    cout << "Stelle Präzission ein..." << endl;
    cout.setf(ios::scientific,ios::floatfield);
    cout.precision(10);
    cout << "f.edge_cell_w=" << f.edge_cell_w << endl;

    os<<f.edge_cell_w;
    os.setf(ios::fixed,ios::floatfield);
    os.precision(2);
    os<<" "<<f.mesh<<flush;
    os.setf(ios::fixed,ios::floatfield);
    return os;
};
```

Die Ausgabe in scientific Darstellung funktioniert nur mit Variablen vom Typ double korrekt. Bei float Variablen werden die hinteren Stellen verstümmelt.

Das folgende Beispielprogramm macht das Problem deutlich:

```
#include <iostream>
#include <iomanip>

using namespace std;

int main(void)
{
  double x = 0.02;
  float y = 0.02;

  cout << setprecision(10) << setiosflags(ios::scientific);
  cout << x << endl;  // Ausgabe: 2.0000000000e-002
  cout << y << endl;  // Ausgabe: 1.9999999553e-002

  cout << "Ok" << endl;
  cin.get();

  return 0;
}
```

Aus diesem Grund wurden die in der Header Datei geoutility.h definierten Strukturtypen dahingehend verändert, dass diese keine float Variablen als Komponenten mehr verwenden.

```
typedef struct
{
    //float z;
    double z;
    Complex epsilon;
    Complex mue;
    Complex sigma;
} geolayer;

typedef struct
{
    //float32 version;
    double version;
    uint32 num_of_comments;
    int32 auto_edge_cell;
    //float32 edge_cell_w;
    double edge_cell_w;
    int32 mesh;

} global_first_line;

typedef struct
{
    //float32 max_freq;
    double max_freq;
    uint32 units;
    int32 f;
    uint32 num_diel_layer;
    uint32 num_metaltype;
    int32 s;
    int32 t;
    //float32 cal;
    double cal;

} global_last_line;
```

Wie sich später herausgestellt hat, war dies nicht das einzige Problem, das zu Rundungsfehlern führte. Ein weiteres Problem trat in ie3dkonverter.cpp auf. Dort wurden folgende, explizite Typumwandlungen vorgenommen:

```
i64_co=(int64)(string2float(zco)*faktor);
```

Leider trat dadurch ein weiterer Rundungsfehler auf. Als Abhilfe habe ich an den entsprechenden Stellen im Programm folgende Änderung vorgenommen:

```
i64_co=double2long(string2float(zco)*faktor);
```

Dazu habe ich globalutilities.cpp um die Funktionen double2long und double2ulong ergänzt:

```
unsigned long double2ulong(double d)
 {
   unsigned long retval = convert(d,retval);
   return retval;
 }

 long double2long(double d)
 {
   long retval = convert(d,retval);
   return retval;
 }
```

Das mag zwar etwas seltsam aussehen, aber damit lässt sich dieses Problem umgehen.
Beide Funktionen basieren wiederum auf der Funktionsschablone convert.

Falsche physikalische Parameter in den dielektrischen Layern

Da die MMICs in Multilayer Technologie gefertigt werden, gibt es leitende Metallschichten und isolierende Schichten, die übereinander gelagert sind. Die physikalischen Parameter der dielektrischen Layer (bzw. der Metallayer) werden in der Technologiedatei angegeben. Hier ein Ausschnitt einer typischen Technologiedatei:

```
######################################
#         dielektrischer Layer
######################################

#--------------------------------------------------------------------
[DielLayer]
          Z                 ==        0
          Er                ==        1
          Ur                ==        1
          Sigma             ==        1e7
          SuperConductor    ==        no

#--------------------------------------------------------------------
[DielLayer]
          Z                 ==        0.7
          Er                ==        7.8tan0.0015
          Ur                ==        1
          Sigma             ==        0
          SuperConductor    ==        no

#--------------------------------------------------------------------
[DielLayer]
          Z                 ==        0.7
          Er                ==        0
          Ur                ==        0
          Sigma             ==        1e7
          SuperConductor    ==        no
```

Das Problem war, dass diese Parameter nicht korrekt in die resultierende geo Datei übernommen wurden.

Die Datensätze für die einzelnen dielektrischen Layer wurden seither vom Konstruktor der Klasse GEODiellayer nach folgendem Schema aus der Technologiedatei gelesen:

1.) Merke Dir die Z Koordinaten aller in der Technologiedatei angegebenen dielektrischen Layer
2.) Sortiere diese Z Koordinaten
3.) Für jede dieser sortierten Z Koordinaten mache folgendes:
4.) Lese die physikalischen Parameter des dielektrischen Layers an dieser Z Koordinate aus der Technologiedatei ein.

Der Fehler lag in der Include Datei geodiellayer.cpp. In einer alten Programmversion wurde offensichtlich ein vector<int> für die Z Koordinaten der dielektrischen Layer verwendet, der in einer späteren Programmversion durch einen vector<double> ersetzt wurde. Dennoch wurde die Funktion int2string auf die Elemente dieses vector angewandt:

```
// Gehe alle Z Koordinaten im vector intzcoord durch...
vector<double>::const_iterator it;  // SE24 changed to vector of double
for(it=intzcoord.begin();it!=intzcoord.end();it++)
{
    //help=Option("Z",int2string(*it),log,dbmode);
    help=Option("Z",double2string(*it),log,dbmode);
```

Dies führte dann dazu, dass Nachkommastellen abgeschnitten wurden. So wurde z.B. aus den Werten 0,0 , 0,7 und 0,7 dreimal der Wert 0. Das war der Grund, warum nur die Parameter des ersten, in der Technologiedatei angegebenen Layers für alle Layer der geo Datei benutzt wurden: Für alle Layer wurden nur die ganzzahligen Anteile der Z Koordinaten übernommen. Und da der Algorithmus versucht, für jede Z Koordinate die physikalischen Parameter des auf dieser Ebene anzutreffenden dielektrischen Layers aus der Technologiedatei zu lesen, schlägt dies fehl, sobald die ganzzahligen Anteile mehrere Z Koordinaten gleich sind.

Die notwendige Funktion double2string, die einen double Wert in einen String schreibt, habe ich der Include Datei globalutilities.cpp hinzugefügt:

```
string double2string(double d)
{
  string retval;
  return convert(d,retval);
}
```

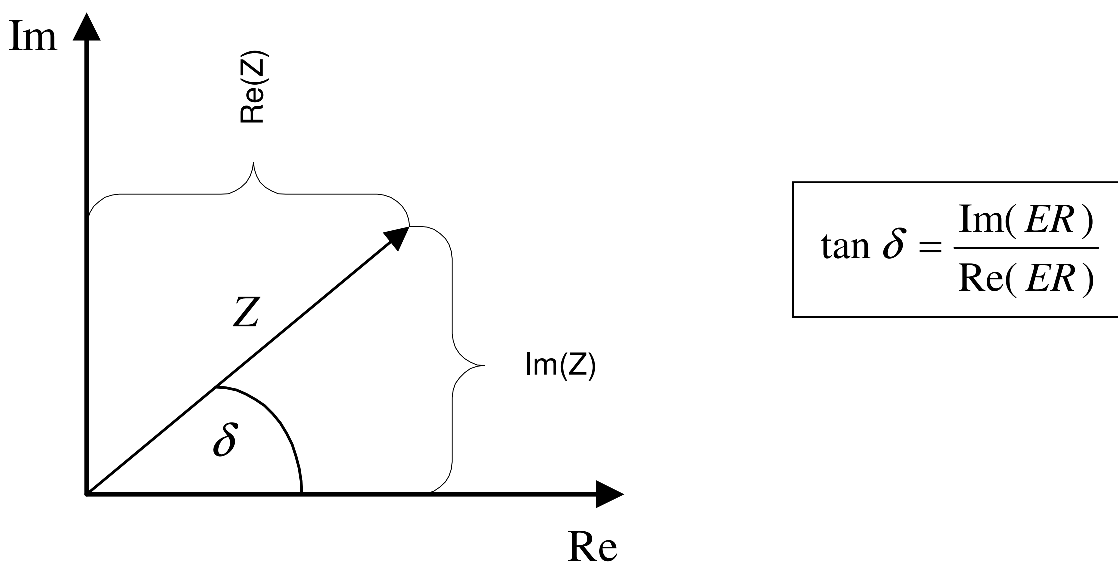
Diese Funktion nutzt wiederum die Funktionsschablone convert.

Der beschriebene Algorithmus zum Einlesen der physikalischen Parameter aus der Technologiedatei versagt jedoch, falls es – wie in unserem Beispiel - mehrere dielektrische Layer mit der selben Z Koordinaten gibt.

Deshalb habe ich den Konstruktor der Klasse GEODiellayer neu implementiert. Die physikalischen Parameter für die dielektrischen Layer werden jetzt einfach der Reihe nach aus der Technologiedatei gelesen, ohne die dielektrischen Layer nach den Z Koordinaten zu sortieren.

Ein weiterer Denkfehler hatte sich in der Behandlung des Falles ER=0 ($\varepsilon_r = 0$) eingeschlichen.

Der Verlustfaktor $\tan\delta$ ergibt sich nämlich aus dem Verhältnis vom Imaginärteil zum Realteil von ER.



$$\tan\delta = \frac{\mathrm{Im}(ER)}{\mathrm{Re}(ER)}$$

Abbildung 8: Eine Komplexe Zahl Z auf der Gaußschen Zahlenebene

Hier müsste man die Quadranten beachten: Für den Fall, dass der Realteil 0 wird gilt: $\tan\delta$ ist $\frac{\pi}{2}$, falls der Imaginärteil größer wie 0 ist, sonst $\frac{3\pi}{2}$. Wenn man diese Betrachtung nicht macht, führt das im Fall von ER=0 zu einer Teilung durch 0.

Für diesen Fall wurde das Programm derart verändert, dass es mit einer Fehlermeldung abbricht, falls in der eingelesenen Technologiedatei ER mit einem Wert kleiner wie 1 angegeben wird, was physikalisch keinen Sinn ergeben würde.

Merke: Der in einer Technologiedatei angegebene Wert für ER muss größer oder gleich 1 sein.

Layer mehrfach verwenden

Bis jetzt war das Programm so ausgelegt, dass in der Technologiedatei angegebene Layer mit eindeutigen Nummern identifiziert wurden. Beim Aufwachsen lassen entstehen aus Polygonen dreidimensionale Körper. Dabei wird für den Deckel des Körpers das selbe Polygon wie für den Boden verwendet, nur mit unterschiedlichen z Koordinaten. Deshalb waren für den Boden und den Deckel eines Körpers jeweils ein separater Layer mit eindeutiger Nummerierung notwendig, wie das folgende Beispiel aus einer Technologiedatei zeigt:

```
#Leit3  Leiter ->GDSLAYER 3 -> Layer number 3
[layer]
GDSLAYER        ==      3
Z               ==      0.28
METALNAME       ==      M
GROW            ==      0

#Leit3  Leiter ->GDSLAYER 63 -> Layer number 3
[layer]
GDSLAYER        ==      63
Z               ==      0.288
METALNAME       ==      M
GROW            ==      0
```

Was hier vorliegt, ist im Prinzip eine Variation des aus der Informatik bekannten *„Pigeonhole Problem"*. Die Polygone aus der gds Datei werden in einer Instanz der Klasse Polygon3Dtable abgelegt. Dort gibt es für jeden Layer aus der gds Datei einen vector. Diese Layer müssen eindeutig nummeriert sein – es darf keine Layernummer zweimal vorkommen.

Die bisher verwendete Definition der Technologiedatei folgte ebenfalls dieser eindeutigen Nummerierung. Es war nicht möglich, eine Layernummer mehrfach anzugeben. Ziel ist es, das Programm so umzuschreiben, dass man eine Layernummer in der Technologiedatei mehrfach mit unterschiedlichen Z Koordinaten und/oder Metalltypen verwenden kann.

Welcher Layer von CADENCE beim gds Export auf welchen Layer in der gds Datei abgebildet wird, wird im einer Mapping Datei festgelegt. In der folgenden Mapping Datei sind die zusätzlichen Layer, die in der neuen Version des Konverters nicht mehr benötigt werden auskommentiert:

```
#name            purpose  no  type
#Leit5           Leiter   6   0
Leit5    Leiter   5  0
#Leit5           Leiter   65  0
Leit4    Leiter   4  0
#Leit4           Leiter   64  0
Leit3    Leiter   3  0
#Leit3           Leiter   63  0
Leit2    Leiter   2  0
#Leit2           Leiter   62  0
#Leit2           Leiter   1   0
#

Leit5    ie3d    10  0
Leit4    ie3d    11  0
Leit3    ie3d    12  0
Leit2    ie3d    13  0

# Via topviews

DiEl2            ie3d       48  0
DiEl3            ie3d       47  0
DiEl4            ie3d       46  0
DiEl5            ie3d       45  0
DiEl6            ie3d       44  0


#
#VIA      ie3d1    11  0
#VIA      ie3d2    12  0
#VIA      ie3d3    13  0
#
#resistor  GA     20  0
#resistor  TN     21  0
```

```
#resistor   OHM     22  0
Wide2               R2          21  0
Wide3               R1          23  0
Wide4               R1          25  0
#
```

Lösungsansatz

Anstatt in der gds bzw. Technologiedatei verwendete Nummerierung zu übernehmen, soll eine eigene, interne Nummerierung der Layer innerhalb des Konverters verwendet werden. Dazu wurde in der Klasse IE3Dkonverter eine laufende Nummer Namens LayerCount eingeführt. Jedem Layer aus der Technologiedatei wird eine solche Nummer zugewiesen. Anschließend wird die Nummer um 1 erhöht. So werden alle Layer intern mit einer laufenden Nummer gekennzeichnet.

Zusätzlich wird in einer sogenannten LayerMap vermerkt, welcher gds Layernummer welche fortlaufende Nummer zugewiesen wurde.

Hier ein Beispiel einer solchen LayerMap:

Bemerkung									TVL	Z	Z	TVL	Z	Z	TVL	Z	Z	TVL	Z	Z	TVL	Z	Z
Gds Layernummer	1	2	3	4	5	6	21	25	48	2	2	47	2	3	46	3	4	45	4	5	44	5	5
Interne Layernummer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

Die Methode RegLayer nimmt diese Eintragung in die LayerMap vor:

```
// Interne Layernummer mit gds Layernummer registrieren
// Rückgabewert ist die interne Nummer für diesen gds Layer
int RegLayer(int gdslnr);
```

Der Rückgabewert von RegLayer ist die interne Nummer, unter welcher der angegebene gds Layer abgelegt wurde. Umgekehrt kann man mit Hilfe der Methode gdslayernr herausfinden, welche gds Layernummer zu einer bestimmten internen Layernummer gehört.

```
// Findet die zur internen Nummerierung der Layer passende
// gds Layer Nr., wie sie ursprünglich in der gds Datei vorhanden war.
int gdslayernr(const int intnr) const;
```

Bisher wurden die gds Layernummern aus der Technologiedatei als Bezeichner innerhalb des Konverters verwendet. Deshalb war es nicht möglich, eine Layernummer mehrfach innerhalb der Technologiedatei zu verwenden, um damit beispielsweise auszudrücken, dass ein Layer einmal für den Boden eines Körpers und – mit unterschiedlichen z Koordinaten - für dessen Deckel verwendet werden soll.

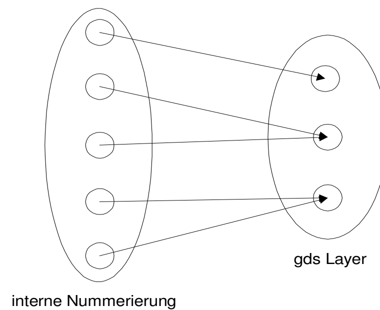
Mit der beschriebenen Abbildung lässt sich das umgehen: Die Bezeichner für die Layer werden vom Konverter intern generiert.

Für folgende Sections aus der Technologiedatei werden interne Layernummern vergeben:

```
[LAYER]
[3DLAYER]
[EXTPORTS]
[VLPLAYER]
```

Außer dem werden den in 3DLAYER Sectios angegebenen Z Layern ebenfalls interne Nummern zugewiesen. So können einer gds Layernummer mehrere interne Nummern zugewiesen werden.
In unserem Beispiel hat der gds Layer 2 die interne Nummer 1. Gleichzeitig bekommt der gds Layer 2 aber auch die Nummer 9,10 und 12 durch die Z Layer zugewiesen.

Die Tabellen laytab ,_3Dlaytab und vlplaytab verwenden nun diese neue Nummerierung, während Instanzen der Klasse Polygon3Dtable weiterhin die in der gds Datei angegebene Nummerierung verwenden.



Die Abbildung ist also surjektiv und nicht injektiv, denn es gibt verschiedene Elemente der internen Nummerierung, die auf die gleichen gds Layer abgebildet werden.

Jedes mal, wenn auf eine Instanz einer Polygon3Dtable zugegriffen wird, wird mit Hilfe der Methode gdslayernr die gds Layernummer zur internen Layernummer ermittelt.

Beispiel:

```
polygone=polytab_[gdslayernr(lnr)];   // indirekter Zugriff über Schlüssel
```

Hier werden alle Polygone eines bestimmten gds Layers angefordert.

Damit ist es nun auch möglich, gds Layer in der Technologiedatei doppelt zu verwenden, wie das folgende Beispiel zeigt:

```
#Leit3  Leiter ->GDSLAYER 3 -> Layer number 3
[layer]
GDSLAYER        ==      3
Z               ==      0.28
METALNAME       ==      M
GROW            ==      0

#Leit3  Leiter ->GDSLAYER 3-> Layer number 3
[layer]
GDSLAYER        ==      3
Z               ==      0.288
METALNAME       ==      M
GROW            ==      0
```

Hier wird der gds Layer Nr. 3 zweimal mit unterschiedlichen Z Koordinaten verwendet.

Ein positiver Seiteneffekt davon ist, dass die gds dadurch Dateien kleiner werden.

Verbesserung des Laufzeitverhaltens

Um das endgültige Programm kleiner und schneller zu machen, wurde ein Schalter eingeführt, mit dem Programmteile, die dem Debugging dienen, bei der Compilierung wahlweise ignoriert werden.
Der Schalter wurde mit Hilfe einer symbolischen Konstanten realisiert.

```
// Neu: Wenn die symbolische Konstante DEBUG gesetzt ist, werden zusätzliche
// Debuggingroutinen mit eingebunden.
// #define DEBUG
```

Wenn die symbolische Konstante DEBUG nicht gesetzt ist, werden bestimmte Programmteile bei der Compilierung nicht in das ausführbare Programm übernommen.

Beispiel:

```
GDSifstream gds_in(gdsf,p_mm,dbmode);          // Abbilden der gds Datei in einen Stream (RAM)
GDSStreamFormat sf(gdsf,p_mm);                 // Erstellen eines "vorübergehendes" StreamFormat Objekt
gds_in>>sf;                                    // Einlesen aller Daten in das StreamFormat Objekt in "roher" Form
#ifdef DEBUG
sf.print(*sendto("gds"));
#endif
polytab_=sf.getPolygontable();                 // aufbauen einer Tabelle in der alle Polygone "ungeordnet" stehen
```

Mit Hilfe der Preprozessordirektiven #ifdef und #endif lassen sich so die Übersetzung bestimmter Programmabschnitte von Bedingungen abhängig machen.

Durch das Weglassen von Debugging Informationen in der Version für den Endanwender konnte so eine Geschwindigkeitssteigerung um etwa Faktor 100 erreicht werden.

Eine weitere Verbesserung konnte beim Herausschreiben der GEO Objekte in die Ausgabedatei durch Verwendung eines Puffers erreicht werden. Die Ausgabe erfolgt zunächst in einen Puffer vom Typ stringstream. Erst wenn alle GEO Objekte im Puffer stehen, wird dieser in den Ausgabestrom geschrieben.

In globalutilities.h und globalutilities.cpp wurden einige häufig verwendete Funktionen inline gemacht, was ebenfalls die Geschwindigkeit erhöht:

```
template <class type_given, class type_wanted>
inline type_wanted convert(type_given &val, type_wanted &typeVar)
{
  stringstream convStream;
  type_wanted var;
  convStream << val << endl; // Wert in den Datenstrom einfügen
  convStream >> var;         // Wert im gewünschten Typ aus dem Datenstrom auslesen
  return var;
};

inline string int2string(int I)
{
   string retval;
   return convert(I,retval);
};

inline string double2string(double d)
{
   string retval;
   return convert(d,retval);
};

inline int string2int(const string &s)
{
  int retval;
  return convert(s,retval);
};

inline double string2float(const string & s)
{
  double retval;
  return convert(s,retval);
};
```

```
inline double string2scientific(const string & s)
{
   double retval;
   return convert(s,retval);
};
```

Ferner werden jetzt keine redundanten Nullen für die Koordinaten der Polygonobjekte in die Ausgabedatei geschrieben. Dies lässt die Dateigröße schrumpfen und erhöht die Geschwindigkeit.

Hier ein Ausschnitt:

Vorher:

```
8 20 1 2 1
-1.8000000000e+00 -1.0000000000e+00 1.4000000000e-01 -1
-1.7750000000e+00 -9.7500000000e-01 1.4000000000e-01 -5
-1.7750000000e+00 -8.2500000000e-01 1.4000000000e-01 -4
-1.6250000000e+00 -8.2500000000e-01 1.4000000000e-01 -3
-1.6250000000e+00 -9.7500000000e-01 1.4000000000e-01 -2
-1.6000000000e+00 -1.0000000000e+00 1.4000000000e-01 -8
-1.6000000000e+00 -8.0000000000e-01 1.4000000000e-01 -6
-1.8000000000e+00 -8.0000000000e-01 1.4000000000e-01 -7
```

Ohne redundante Nullen:

```
8 20 1 2 1
-1.8 -1 0.14 -1
-1.775 -0.975 0.14 -5
-1.775 -0.825 0.14 -4
-1.625 -0.825 0.14 -3
-1.625 -0.975 0.14 -2
-1.6 -1 0.14 -8
-1.6 -0.8 0.14 -6
-1.8 -0.8 0.14 –7
```

Da es sich um ASCII Dateien handelt, spart man so je weggelassenes Zeichen 1 Byte, was sich insbesondere bei größeren Dateien bemerkbar macht.

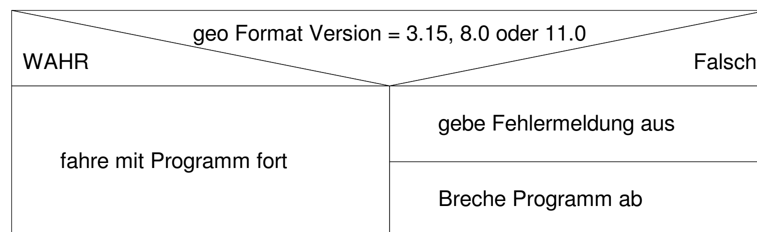
Implementierung des neuen IE3D XML 2.0 Formats

Mit der Version 11.0 von IE3D wurde das neue IE3D XML 2.0 Format für die geo Dateien eingeführt.

Dieses neue Format galt es nun für den Konverter zu implementieren. Dabei sollte wahlweise auch das alte geo Format ausgegeben werden können. Die Auswahl, welches geo Format ausgegeben werden soll, erfolgt dabei durch Angabe eines Parameters in der Technologiedatei.

```
[Global]
        formatversion       ==      11.0
        CommentLine         ==      NoComment
        Commentline         ==      nocomment
        AutoEdgeCell        ==      No
        EdgeCell            ==      2.0e-2
        MaxFreq             ==      40
        Cal                 ==      1.5e+3
        Cells               ==      20
        TopEnclosure        ==      No
        BottomEnclosure     ==      No
        LeftEnclosure       ==      no
        RightEnclosure      ==      no
        Plnwave   ==        no
```

In der Section Global kann man durch Setzen der Option formatversion die gewünschte Version der geo Ausgabedatei wählen. Erlaubt sind die Werte 3.15, 8.0 oder 11.0. Bei anderen Werten bricht der Konverter mit einer Fehlermeldung ab.



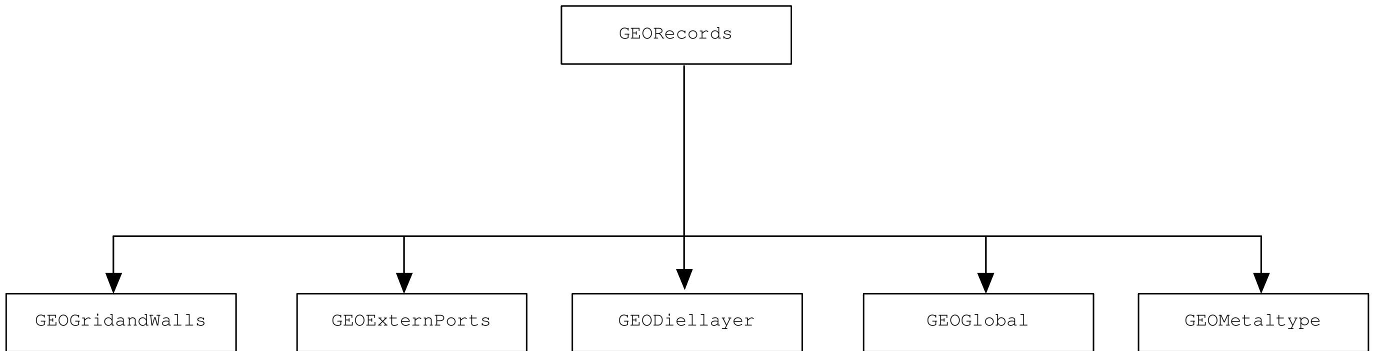
Die Implementierung habe ich innerhalb von gds2ie3d.cpp (wo sich main befindet) platziert:

```
// Überprüfung, ob angegebene Zahl gültig ist
if (! ((geoversion == 3.15) || (geoversion == 8.0) || (geoversion == 11.0)) )
{
  *mm.sendto("konverter") << "Fehler! Die in der Technologiedatei angegebene Versionsnummer ";
  *mm.sendto("konverter") << double2string(geoversion) << " für die geo Datei ist nicht gültig." << endl;
  *mm.sendto("konverter") << "Gültige Versionsnummern sind 3.15, 8.0 und 11.0." << endl;
  exit(EXIT_FAILURE);
}
```

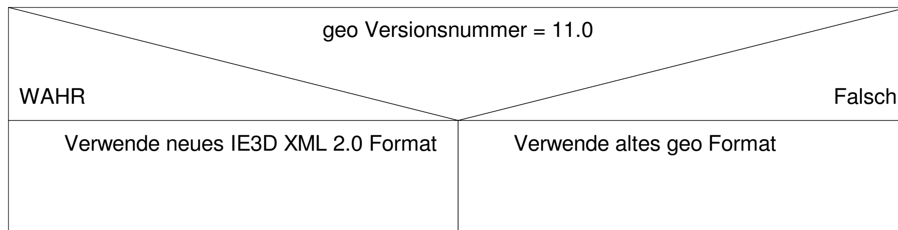
Um innerhalb der GEO Klassen den Parameter formatversion aus der Technologiedatei zu ermitteln, habe ich in geoutility.h eine neue inline Funktion Names GetGEOVersion eingeführt:

```
// Versionsnummer für die geo Datei aus der Technologiedatei ermitteln
inline double GetGEOVersion(const Parser *p)
{
  // p ist Zeiger auf Instanz der Parser Klasse
  return p->getSectionContainer("GLOBAL").getSection(0).getOption("FORMATVERSION").getvalueAsfloat();
};
```

Für die Ausgabe der geo Datei existiert im Konverter ein Klassensystem:



Jede dieser Klassen repräsentiert dabei einen Teil der geo Datei. Zum Schreiben in den Ausgabestrom besitzen diese Klassen jeweils eine Methode namens print. Diese wurde von mir so verändert, dass jetzt wahlweise das neue oder das alte geo Dateiformat ausgegeben werden kann.



Das folgende Beispiel stellt die Implementierung der Methode print für die Klasse GEOGlobal dar:

```
void GEOGlobal::print(ostream & os) const
{
        double VersNo = GetGEOVersion(p);

        if (VersNo==11.0) // IE3D XML 2.0 Format, bitte!
        {
         int32  AEC      = first.auto_edge_cell;
         string UNITS    = UnitString(last.units);
         string MAXFREQ  = double2string(last.max_freq);
         string EPSLENGTH = "1e-006";

         os << "<?xml version=\"1.0\" ?>" << endl
            << "<ZlsDoc type=\"Ie3dSim xml file\" version=\"2.0\""
            << " copyright=\"Zeland Software, Inc.\">" << endl
            << "<Ie3dSim unit=\"" << UNITS << "\""
            << " version_str=\"11.0\" final_ports=\"0\" license_version=\"0\">" << endl
            << "<!-- Ie3dSim base type begin -->" << endl
            << "<Ie3dGeom format_version=\"20.1\" comments=\"\""
            << " eps_length=\"" << EPSLENGTH << "\""
            << " netports=\"0\" syn_layers=\"0\" nbr=\"0\" scheme_deembed=\"1\""
            << " nhext_mmic=\"5\" nhnext_wave=\"5\" Bmetal=\"0\""
            << " Bgroundconnect=\"1\" Bshowextent=\"1\" Bsavecurrent=\"0\""
            << " Bkeepdiscretize=\"1\" Bshowvertex=\"0\" dc_mode_index=\"0\">" << endl;

         // Meshing Parameter
         os << "<!-- Ie3dGeom base type begin -->" << endl
            << "<MeshingParametersBase ereff=\"5.45000247499982\""
            << " fmax=\"" << MAXFREQ << "\""
            << " ncells=\"20\" warning_limit=\"4000\" aec=\"" << AEC << "\""
            << " aec_level=\"0\" aec_ratio=\"0.1\" multi_aec_ratio=\"0.4\""
            << " meshing_optim=\"1\" detect_overlapping=\"1\""
            << " meshing_scheme=\"0\" align_meshing=\"4\""
            << " max_layer_distance=\"0.0005\" cmax_regular=\"0.3210423766983152\""
            << " refined_ratio=\"0.2\" rectanglizations=\"3\" merge_polygons=\"0\""
            << " option_2d=\"2\" option_3d=\"2\">" << endl
            << "</MeshingParametersBase>" << endl
```

```
        << "<!-- Ie3dGeom base type end -->" << endl;
      }
      else // altes geo Format
      {
       os<<first<<endl;

       vector<string>::const_iterator it;

       for(it=comments.begin();it!=comments.end();it++)
       {
           os<<*it<<endl;
       };

       os<<last<<endl;
      }
};
```

Hier ein Beispiel der XML Ausgabe der Klasse GEOGlobal:


```
<?xml version="1.0" ?>
<ZlsDoc type="Ie3dSim xml file" version="2.0" copyright="Zeland Software, Inc.">
<Ie3dSim unit="mm" version_str="11.0" final_ports="0" license_version="0">
<!-- Ie3dSim base type begin -->
<Ie3dGeom format_version="20.1" comments="" eps_length="1e-006" netports="0" syn_layers="0" nbr="0" scheme_deembed="1"
nhext_mmic="5" nhnext_wave="5" Bmetal="0" Bgroundconnect="1" Bshowextent="1" Bsavecurrent="0" Bkeepdiscretize="1" Bshowvertex="0"
dc_mode_index="0">
<!-- Ie3dGeom base type begin -->
<MeshingParametersBase ereff="5.45000247499982" fmax="40" ncells="20" warning_limit="4000" aec="0" aec_level="0" aec_ratio="0.1"
multi_aec_ratio="0.4" meshing_optim="1" detect_overlapping="1" meshing_scheme="0" align_meshing="4" max_layer_distance="0.0005"
cmax_regular="0.3210423766983152" refined_ratio="0.2" rectanglizations="3" merge_polygons="0" option_2d="2" option_3d="2">
</MeshingParametersBase>
<!-- Ie3dGeom base type end -->
```

Backus Naur Form des GDS Formats

<table>
<tr><td><stream format></td><td>::=</td><td>HEADER BGNLIB LIBNAME [REFLIBS] [FONTS]
[ATTRTABLE] [GENERATIONS] [<FormatType>]
UNITS {<structure>}* ENDLIB</td></tr>
<tr><td><FormatType></td><td>::=</td><td>FORMAT | FORMAT {MASK} + ENDMASKS</td></tr>
<tr><td><structure></td><td>::=</td><td>BGNSTR STRNAME [STRCLASS] {<element>}* ENDSTR</td></tr>
<tr><td><element></td><td>::=</td><td>{<boundary> | <path> | <sref> | <aref> | <text> | <node> | <box>} {<property>}* ENDEL</td></tr>
<tr><td><boundary></td><td>::=</td><td>BOUNDARY [ELFLAGS] [PLEX] LAYER DATATYPE XY</td></tr>
<tr><td><path></td><td>::=</td><td>PATH [ELFLAGS] [PLEX] LAYER DATATYPE [PATHTYPE][WIDTH] XY</td></tr>
<tr><td><sref></td><td>::=</td><td>SREF [ELFLAGS] [PLEX] SNAME [<strans>] XY</td></tr>
<tr><td><aref></td><td>::=</td><td>AREF [ELFLAGS] [PLEX] SNAME [<strans>] COLROW XY</td></tr>
<tr><td><text></td><td>::=</td><td>TEXT [ELFLAGS] [PLEX] LAYER <textbody></td></tr>
<tr><td><node></td><td>::=</td><td>NODE [ELFLAGS]. [PLEX] LAYER NODETYPE XY</td></tr>
<tr><td><box></td><td>::=</td><td>BOX [ELFLAGS] [PLEX] LAYER BOXTYPE XY</td></tr>
<tr><td><textbody></td><td>::=</td><td>TEXTYPE [PRESENTATION] [PATHTYPE] [WIDTH] [<strans>] XY STRING</td></tr>
<tr><td><strans></td><td>::=</td><td>STRANS [MAG] [ANGLE]</td></tr>
<tr><td><property></td><td>::=</td><td>PROPATTR PROPVALUE</td></tr>
</table>

Portierung auf gcc (g++)

Dinge, die eine Portierung des Konverters behinderten

- In gds2ie3d.cpp hier wird Borland spezifische Anweisung USEUNIT verwendet.
- In globalutilities.cpp: Hier wird x86 Assembler Code verwendet.
- Diverse Typecast Fehler

Ersetzen der Assembler-Sequenzen

In globalutilities.h und globalutilities.cpp war folgende Assembler Code Sequenz enthalten:

```
unsigned short int high(unsigned long a)
 {
  unsigned short int back;

  asm{
       push  eax
       mov   eax,[a+2]
       mov   back,ax
       pop   eax
     }
   return back;
 }
```

Da der gcc ein anderes Format für Assembler Sequenzen als der Borland Compiler verwendet, musste diese Sequenz ersetzt werden. Dabei habe ich mich dafür entschieden, anstatt Assembler Code lieber Lowlevel C Code zu verwenden:

```
inline unsigned short int high(unsigned long a)
{
   (int)(a>>16);        // highword holen
};
```

Dies funktioniert allerdings nur bei LittleEndian Systemen, wie z.B. der x86 Architektur, bei denen der Datentyp long genau 4 Byte breit ist. Für eine Portierung des Programms auf UltraSparc Solaris muss dieser Codeabschnitt erneut angepasst werden.

Die neue Timer Klasse

Da das alte Zeitmessungssystem nicht ANSI konforme Funktionen und globale Variablen verwendete, habe ich mich entschlossen, eine neue Timer Klasse zu schreiben.

Diese einfache Timer Klasse funktioniert wie eine Stopuhr: Die Methode Start() startet die Zeitmessung, die Methode Stop() beendet die Zeitmessung, Reset() setzt die Zeit auf 0 zurück und Difftime() gibt die gemessene Zeit in Sekunden zurück.

Der wesentliche Vorteil der Timer Klasse ist, dass man verschiedene Instanzen der Timer Klasse haben kann, die dann jeweils unabhängig voneinander (und quasi nebenläufig) individuelle Zeitintervalle messen können.

```
//  Timer.h
//  Eine Klasse zur Zeitmessung
//  Autor: Norman Walter

#ifndef TIMER_H
#define TIMER_H

#include <ctime>

class Timer
{
    // Attribute
    private:
        clock_t starttime;
        clock_t endtime;

    public:

    // Konstruktor
    Timer(void)
    {
      starttime = 0;
      endtime   = 0;
    }

    // Destruktor
    ~Timer(void)
    {
    }

    // Methoden

    // Zeitmessung starten
    void Timer::Start(void);

    // Zeitmessung stoppen
    void Timer::Stop(void);

    // Zeitmessung Zurücksetzen
    void Timer::Reset(void);

    // Die gemessene Zeit zurückgeben
    double Timer::Difftime(void);

};

#endif
```

Makefile für gds2ie3d

Für gds2ie3d wurden mehrere Makefiles erstellt. Die integrierte Entwicklerumgebung Dev C++ für Microsoft Windows erzeugt ihre eigenen Makefiles automatisch. Für die Betriebssysteme x86 Linux und Sparc64 Solaris wurden entsprechende Makefiles für gmake (GNU Make) von Hand geschrieben.

Als Beispiel folgt das Makefile für Sparc64 Solaris:

```
# Makefile für gds2ie3d (Solaris Version)
# Autor: Norman Walter
# Datum: 30.11.2005

CC=g++
CFLAGS=-c -O3
LDFLAGS=-ansi -static
BASEPATH=/user/nw82bk/Konverter_gcc/SourceCode/
INCS = -I"$(BASEPATH)code" -I"$(BASEPATH)code/3dpolygone" \
-I"$(BASEPATH)code/gds_einlesen" -I"$(BASEPATH)code/global" \
-I"$(BASEPATH)code/tek_einlesen" -I"$(BASEPATH)code/ErrorHandler" \
-I"$(BASEPATH)code/manager" -I"$(BASEPATH)code/tables" \
-I"$(BASEPATH)code/geo_erzeugen" -I"$(BASEPATH)code/konverter"
SOURCES=gds2ie3d.cpp $(BASEPATH)code/3dpolygone/coord3d.cpp \
$(BASEPATH)code/3dpolygone/edge.cpp \
$(BASEPATH)code/3dpolygone/polygon3D.cpp \
$(BASEPATH)code/3dpolygone/coord2d.cpp \
$(BASEPATH)code/konverter/parameter.cpp \
$(BASEPATH)code/konverter/polygoncontainer.cpp \
$(BASEPATH)code/konverter/ie3dkonverter.cpp \
$(BASEPATH)code/manager/messagemanager.cpp \
$(BASEPATH)code/ErrorHandler/ErrorHandler.cpp \
$(BASEPATH)code/gds_einlesen/boundaryelement.cpp \
$(BASEPATH)code/gds_einlesen/elementcontainer.cpp \
$(BASEPATH)code/gds_einlesen/gdsbgnlib.cpp \
$(BASEPATH)code/gds_einlesen/gdsbgnstr.cpp \
$(BASEPATH)code/gds_einlesen/gdsboundary.cpp \
$(BASEPATH)code/gds_einlesen/gdsdatatype.cpp \
$(BASEPATH)code/gds_einlesen/gdsdate.cpp \
$(BASEPATH)code/gds_einlesen/gdsendel.cpp \
$(BASEPATH)code/gds_einlesen/gdsendlib.cpp \
$(BASEPATH)code/gds_einlesen/gdsendstr.cpp \
$(BASEPATH)code/gds_einlesen/gdsheader.cpp \
$(BASEPATH)code/gds_einlesen/gdsifstream.cpp \
$(BASEPATH)code/gds_einlesen/gdslayer.cpp \
$(BASEPATH)code/gds_einlesen/gdslibname.cpp \
$(BASEPATH)code/gds_einlesen/gdspath.cpp \
$(BASEPATH)code/gds_einlesen/gdspathtype.cpp \
$(BASEPATH)code/gds_einlesen/gdspolygon.cpp \
$(BASEPATH)code/gds_einlesen/gdsrecord.cpp \
$(BASEPATH)code/gds_einlesen/gdsreflibs.cpp \
$(BASEPATH)code/gds_einlesen/gdsstreamformat.cpp \
$(BASEPATH)code/gds_einlesen/gdsstrname.cpp \
$(BASEPATH)code/gds_einlesen/gdsunits.cpp \
$(BASEPATH)code/gds_einlesen/gdsutility.cpp \
$(BASEPATH)code/gds_einlesen/gdswidth.cpp \
$(BASEPATH)code/gds_einlesen/gdsxy.cpp \
$(BASEPATH)code/gds_einlesen/pathelement.cpp \
$(BASEPATH)code/gds_einlesen/recordheader.cpp \
$(BASEPATH)code/gds_einlesen/structurecontainer.cpp \
$(BASEPATH)code/gds_einlesen/structureelement.cpp \
$(BASEPATH)code/geo_erzeugen/geoextports.cpp \
$(BASEPATH)code/geo_erzeugen/geoglobal.cpp \
$(BASEPATH)code/geo_erzeugen/geodiellayer.cpp \
$(BASEPATH)code/geo_erzeugen/georecord.cpp \
$(BASEPATH)code/timer/timer.cpp \
$(BASEPATH)code/tables/table_int_uint.cpp \
$(BASEPATH)code/tables/porttable2.cpp \
$(BASEPATH)code/tables/table_double_uint.cpp \
$(BASEPATH)code/tables/table_int_int.cpp \
$(BASEPATH)code/tables/table_int_udouble.cpp \
$(BASEPATH)code/tables/polygon3dtable.cpp \
$(BASEPATH)code/tek_einlesen/sectioncontainer.cpp \
$(BASEPATH)code/tek_einlesen/parser.cpp \
$(BASEPATH)code/tek_einlesen/parserutilities.cpp \
$(BASEPATH)code/tek_einlesen/section.cpp \
$(BASEPATH)code/tek_einlesen/option.cpp \
```

```
$(BASEPATH)code/global/types.cpp \
$(BASEPATH)code/global/globalutilities.cpp \
$(BASEPATH)code/global/mycomplex.cpp \
$(BASEPATH)code/global/baseobject.cpp \
$(BASEPATH)code/geo_erzeugen/polygongeo.cpp \
$(BASEPATH)code/geo_erzeugen/EdgeGEO.cpp \
$(BASEPATH)code/geo_erzeugen/geogridandwalls.cpp \
$(BASEPATH)code/geo_erzeugen/geometaltype.cpp \
$(BASEPATH)code/geo_erzeugen/geoutility.cpp \
$(BASEPATH)code/geo_erzeugen/EdgeContainer.cpp \

OBJECTS=$(SOURCES:.cpp=.o)

EXECUTABLE=gds2ie3d_sparc64_solaris

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE) : $(OBJECTS)
	$(CC) $(LDFLAGS) $(INCS) $(OBJECTS) -o $@

.cpp.o:
	$(CC) $(CFLAGS) $(INCS) $< -o $@

clean :
	rm $(EXECUTABLE) $(OBJECTS)
```

Fazit

Der Konverter verschafft dem Unternehmen einen erheblichen Wettbewerbsvorteil, da Entwürfe, die für die Produktion angefertigt werden, so direkt in den Feldsimulator übernommen werden können, ohne dass jemand ein zweites Modell anfertigen muss. Dadurch wird enorm viel Zeit bei der Entwicklung der Schaltungen eingespart. Außerdem wird so eine Konsistenz der Daten erreicht – man muss nicht zwei Datensätze (2D- und 3D Modell) parallel pflegen.

Dadurch, dass der Konverter jetzt wesentlich schneller läuft (etwa Faktor 100), kann man auch bequem relative große Schaltungen, wie z.B. ein komplettes RF-Modul in nur wenigen Sekunden konvertieren. Wenn man bedenkt, dass der Entwickler seine Schaltung immer wieder ändert, neu simuliert und nochmals nachbessert, solange bis alles stimmt, wird deutlich, wie wichtig es ist, dass die Erzeugung eines 3D Modells möglichst wenig Zeit raubt.

Zeitplan

ID	Vorgangsname	Anfangsdatum	Enddatum	Dauer
1	Einarbeitung	01.09.2005	20.09.2005	14t
2	"Deckungsgleiche Polygone" : Analyse und Planung	08.09.2005	09.09.2005	2t
3	"Deckungsgleiche Polygone" : Implementierung	10.09.2005	13.09.2005	2t
4	"Rundungsfehler" : Analyse und Planung	13.09.2005	19.09.2005	5t
5	"Rundungsfehler" : Implementierung	19.09.2005	20.09.2005	2t
6	"Physikalische Parameter" : Analyse und Planung	20.09.2005	21.09.2005	2t
7	"Physikalische Parameter" : Implementierung	21.09.2005	22.09.2005	2t
8	"Layer" : Analyse und Planung	22.09.2005	29.09.2005	6t
9	"Layer" : Implementierung	27.09.2005	05.10.2005	7t
10	"XML Export" : Analyse und Planung	05.10.2005	20.10.2005	12t
11	"XML Export" : Implementierung	08.10.2005	20.10.2005	9t
12	"Portierung auf GNU Compiler" : Analyse und Planung	20.10.2005	07.11.2005	13t
13	"Portierung auf GNU Compiler" : Implementierung	21.10.2005	07.11.2005	12t
14	"Portierung auf Solaris und Linux" : Analyse und Planung	07.11.2005	01.12.2005	19t
15	"Portierung auf Linux und Solaris" : Implementierung	08.11.2005	01.12.2005	18t
16	Test	09.09.2005	23.12.2005	73t

Meilensteine

13. September 2005:

Deckungsgleiche Polygone werden jetzt nicht mehr in die geo Datei übernommen.

20. September 2005:

Problem „Rundungsfehler“ gelöst

22. September 2005:

Physikalische Parameter werden jetzt korrekt aus der Technologiedatei übernommen.

5. Oktober 2005:

Die Abbildung der gds Layer auf die interne Nummerierung wurde erfolgreich getestet. Damit ist es nun möglich einen gds Layer z.B. für den Boden und den Deckel eines Körpers zu verwenden.

20. Oktober 2005:

Der optionale Export im neuen IE3D XML 2.0 Format funktioniert jetzt.

7. November 2005:

Der Code ist jetzt auch mit dem gcc (bzw. g++) kompatibel.

1. Dezember 2005:

Portierung auf x86 Linux und Sparc64 Solaris abgeschlossen.

Literaturverzeichnis

Development and verification of an automated 2-d to 3-d
file converter for a method of moments program
von Nuria Carrasco Comes
TU Braunschweig, Februar 2000

Praktikumsbericht 2. Praxissemester
von Nina Röhm
Marconi Communications GmbH Backnang

IE3D User's Manual
von Zeland Software, Inc.

Programmieren in C
von Brian W. Kernighan und Dennis M. Ritchie
ISBN 3-446-15497-3

Softwaretechnik in C und C++
von Rolf Isernhagen
ISBN 3-4461-8201-2

C kurz & gut
von Peter Prinz und Ulla Kirch-Prinz
ISBN 3-89721-238-2

Algorithmen in C
von Robert Sedgewick
ISBN 3-8273-7182-1

C++ Der schnelle Einstieg
von Peter Wollschlaeger
ISBN 3-8272-6501-0

Die C++ Programmiersprache
von Bjarne Stroustrup
ISBN 3-8273-2058-5

Einführung in die Automatentheorie,
Formale Sprachen und Komplexitätstheorie
von John E. Hopcroft, Rajeev Motwani und Jeffrey D. Ullman
ISBN 3-8273-7020-5

Halbleiter
Technische Erläuterungen, Technologien und Kenndaten
ISBN 3-89578-205-X

Objektorientiertes Modellieren und Entwerfen mit UML
Skript zum Gastvortrag an der Fachhochschule für Technik Esslingen
Von Prof. Dr. József Tick, Technische Hochschule Budapest

Elektronik für Ingenieure und Naturwissenschaftler
von Ekbert Hering, Klaus Bressler und Jürgen Gutekunst
ISBN 3-540-24309-7