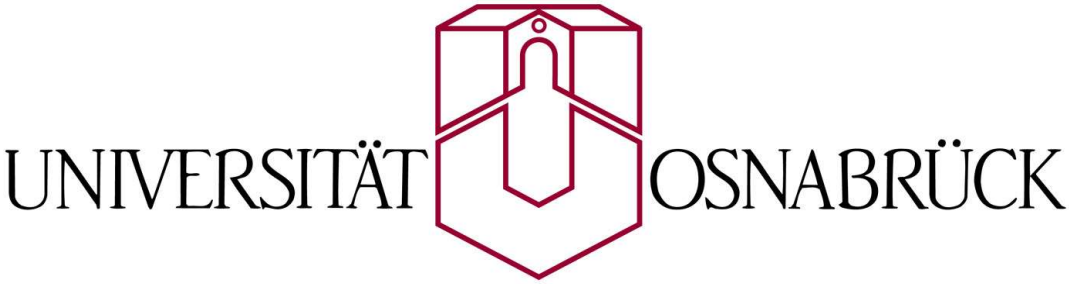



# Bachelorarbeit

---

## Thema:

## Die Osnabrücker Altstadt als VRML-Welt

---

Verfasser:
Katrin Töpler

Gutachter:
Prof. Dr. Oliver Vornberger
Juniorprof. Dr. Sigrid Knust

Fachbereich Mathematik/Informatik
Universität Osnabrück

Abgabetermin:
21.10.2005

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Quelltextverzeichnis

# Danksagung

Ich möchte mich bei den Menschen bedanken, die mich während dieser Bachelorarbeit unterstützt haben. Bestimmten Personen gilt besonderer Dank.

Ich danke Herrn Prof. Dr. Oliver Vornberger für die gute Betreuung der Arbeit im Allgemeinen.
Für die Übernahme des Zweitgutachtens bedanke ich mich bei Frau Juniorprof. Dr. Sigrid Knust.
Herrn Ralf Kunze möchte ich mich für seine wichtigen Hinweise und Anregungen danken.
Für die technische Unterstützung danke ich Friedhelm Hofmeyer.
Ein besonders herzlicher Dank geht an meine Familie, Freunde und Kommilitonen, die mir zu jeder Zeit deren Rückhalt und Unterstützung anboten.

# I. Grundlagen

# 1 Einleitung

In der heutigen Zeit ist das World Wide Web in nahezu allen Haushalten zu erreichen. Damit ist für Unternehmen eine Präsenz im Internet jeglicher Art unverzichtbar. Auch die Stadt Osnabrück ist daran interessiert ihre Internetpräsenz zu verschönern.
Nicht nur Osnabrücker Bürger informieren sich im WWW über Osnabrück, sondern auch andere Personengruppen wie zum Beispiel Touristen. Informationen über Geschäfte, die Geschichte und die Ereignisse der Stadt in Form von Texten und Bildern bietet jede Homepage einer Stadt. Um aber Touristen oder Ortsfremden die Stadt nahe zubringen, ist es von besonderer Bedeutung, die Stadt mittels einer 3D-Welt darzustellen. So können sich Bürger der Stadt und potentielle Touristen leicht und realitätsbezogen über Osnabrück informieren. Sie können virtuell über den Rathausplatz schlendern, die Marienkirche betrachten und Informationen über das Rathaus, die Stadtbibliothek oder die Touristinformation einholen. Die Webpräsenz wird erheblich verstärkt, weil Benutzer einen neuen Anreiz finden, die Internetseite der Osnabrücker Stadt zu besuchen. Sie treffen so beim Gang durch ihre virtuelle Stadt auf neue Informationen wie Werbung oder Ankündigungen neuer Ereignisse. Schließlich macht ein Internetauftritt, mit der Möglichkeit einen 3-dimensionalen Stadtrundgang zu erleben, die Stadt Osnabrück attraktiver.

## 1.1 Aufgabenstellung

Der Grundstein der Aufgabe besteht darin, eine virtuelle 3D-Welt für einen Teil der Osnabrücker Altstadt zu erstellen. Neben dem Aufbau der einzelnen Häuser sollen bestimmte Features eingebaut werden. Zunächst ist interaktiver Text erwünscht, sodass Informationen über die Geschichte der Stadt oder Werbung für anliegende Unternehmen bzw. Geschäfte angezeigt werden können. Weitere Features sind wünschenswert, aber nicht zwingend erforderlich für eine gelungene 3D-Welt. So

soll der Benutzer in bestimmte Gebäude, zum Beispiel in die Marienkirche, in den Dom oder ins Rathaus gehen können, um sich im Inneren zum einen die Gebäude anzusehen und zum anderen Informationen über Geschichte oder Architektur einholen zu können. Außerdem ist vorgesehen, dass, wenn sich mehrere Benutzer in der Welt befinden, diese untereinander kommunizieren können.

Des Weiteren dient diese Bachelorarbeit für die Stadt Osnabrück als Versuchsobjekt mit der Absicht, den Aufwand abzuschätzen eine 3D-Welt für die ganze Innenstadt zu erstellen.

Die Aufgabenstellung hat sich im Laufe der Zeit dahingehend verändert, dass eine Applikation erstellt wird, mit der es ermöglicht, wird eine 3D-Stadt zu erstellen.

Hierbei wird es dem Benutzer ermöglicht, verschiedene Daten einzugeben, und mit wenig Aufwand verschiedene Objekte wie verschiedene Häuserformen oder interaktiven Text anzulegen. Dieses erleichtert die Erstellung eines Grundbaus einer 3D-Stadt enorm. Es stellt sich die Frage, inwieweit es sinnvoll ist Objekte vorzugeben. Denn je komplizierter die angelegten Objekte werden, desto mehr Möglichkeiten der Darstellung sind gegeben und desto mehr Eingabedaten werden benötigt. Letztendlich können komplizierte VRML-Features, wie das Aufeinandertreffen zweier Betrachter, die sich zur gleichen Zeit in der Welt befinden, aufgrund der Anzahl der verschiedenen Darstellungsmöglichkeiten nicht immer zufrieden stellend implementiert werden. Eine Applikation, die die Erstellung des Grundbaus einer 3D-Stadt zur Aufgabe hat, muss also Kompromisse bei der Funktionalität und Anzahl der Features eingehen.

## 1.2 Aufbau der Arbeit

Die Ausarbeitung ist in drei Abschnitte gegliedert.

Der erste Teil beschäftigt sich mit dem verwendeten Programmen und Techniken. Es wird der 3D-Modellierer Vizx3D vorgestellt. Weiterhin werden die wichtigsten verwendeten Grundelemente von VRML beschrieben.

Im zweiten Abschnitt liegt der Fokus auf der im Laufe der Bachelorarbeit entstandenen Applikation. Hier werden anhand von Besonderheiten und Problemen bei der Programmierung einige Vor- und Nachteile von VRML und PHP beleuchtet.
Im letzten Teil wird ein Fazit gezogen und ein kurzer Ausblick auf die noch erweiterbare Applikation gegeben.
Die entstandene Applikation ist unter
https://snowball.informatik.uni-osnabrueck.de/ktoepler/Bachelor/
zu erreichen.

# 2 Verwendete Programme und Programmiertechniken

## 2.1 VRML 97

### 2.1.1 Versionsrückblick

Die erste Version von VRML (Virtual Reality Modeling Language), VRML 1.0, wurde in den Jahren 1994 und 1995 entwickelt, gefolgt von der stark weiterentwickelten inoffiziellen Nachfolger-Version 2.0, die 1996 dem ISO Komitee vorgestellt wurde. Die Version 2.0 beschränkte sich nicht mehr nur auf statische Welten, sondern erlaubte auch Animationen und Benutzerinteraktionen. Ein weiteres Jahr später erklärte die ISO [ISO-STD], diese VRML-Version nach kleineren Änderungen zum internationalen Standard mit dem Namen „VRML 97“ [VRML9701].

### 2.1.2 Grundausrüstungen für den Benutzer

VRML Dateien lassen sich, wie HTML-Dateien, einfach mit einem Texteditor erstellen. Dies hat den Vorteil, dass man den Quelltext vollkommen kontrollieren kann.
Alternativ kann auch ein 3D-Modellierer mit VRML-Konverter verwendet werden, der die Code-Erzeugung übernimmt und so das Entwickeln speziell von komplexen Szenen vereinfacht. Nachteilig ist dabei, dass der erzeugte Quelltext oft nicht optimal ist, was zu größeren Dateien führt. Außerdem unterstützen die meisten dieser Programme nicht den vollen Funktionsumfang von VRML.

Der Quelltext wird clientseitig von einem Browser-Plugin interpretiert und dargestellt. Das CosmoPlayer-Plugin und das blaxxun Contakt-Plugin eignen sich gut zur Darstellung von 3D-Welten. Des Weiteren wurde das blaxxun Contakt-Plugin verwendet, das eine Vielzahl von Features beinhaltet. Vorteile des blaxxun Contact sind unter anderem, die Verwendung als Java-Applet und eine erhöhte Unterstützung der Grafikkarte. Die Navigation ist somit flüssiger als bei dem CosmoPlayer. Anders als beim CosmoPlayer sind beim blaxxun Contact die Bedienungsanleitungen auf Deutsch und es unterstützt die Multi-User-Fähigkeit. Das blaxxun Contakt-Plugin besitzt ein etwas gewöhnungsbedürftiges Benutzerinterface im Vergleich zu anderen Plugins, da es keine Navigationskonsole gibt. Alle Optionen und Funktionen sind nur mit der rechten Maustaste zu erreichen. Das Plugin des blaxxun Contact ist bei den Firmen Blaxxun (http://www.blaxxun.com) und Bitmanagement Software GmbH (http://www.bitmanagement.de) erhältlich.

### 2.1.3 Eine Übersicht von VRML

Die folgenden Kapitel enthalten eine kurze Übersicht über die wichtigsten VRML-Inhalte, die in dieser Arbeit verwendet werden.

#### 2.1.3.1Die Grundstruktur einer VRML-Datei

VRML-Welten werden im Klartext in eine Datei mit „wrl“-Endung abgelegt, die stets mit `#VRML` und der Versionskennung beginnt, beispielsweise `#VRML V2.0 utf8`. Mittels des `<EMBED>` -Tags lässt sich eine 3D-Szene in eine HTML-Seite einbinden. So wird wie bei HTML die Plattformunabhänigkeit der Sprache garantiert.

Der Aufbau einer 3D-Welt wird durch einen Graphen, der die hierarchische Struktur der 3D-Welt angibt, beschrieben. Dieser Graph wird Szenegraph (engl.: `scene graph)` genannt, in ihm werden Objekte und deren Eigenschaften

definiert. Knoten (englisch: `Nodes`) geben die Instanzen des Graphen an. Ein Knoten kann einer Box, einer Lichtquelle oder einer Gruppe entsprechen. Knoten speichern ihre Daten in `fields` ab.
Der Szenegraph ist ein gerichteter, azyklischer Graph. Knoten können also andere Knoten enthalten und dürfen auch selbst in mehreren Knoten enthalten sein. Aber ein Knoten darf sich nicht selbst enthalten. Diese Art der Struktur erleichtert es, eine komplizierte Welt aus mehreren Einzelstücken anzufertigen. Ein Haus kann also mit mehreren Wänden beschrieben werden, die zu einer Gruppe von `IndexedFaceSets` zusammengefasst werden [VRML9702].

### 2.1.3.2 Besondere Ausdrücke

a) `Events` und `Routes`

Das wichtigste Feature seit VRML 2.0 sind `Events`. Durch `Events` wird die Welt beweglich gemacht, denn um etwas in einer VRML-Welt ändern zu können, muss ein `Event` an den betreffenden Knoten geschickt werden. Sie sind die Grundlage für alle Animationen oder Interaktionen. So können die Knoten eines Szenegraphen untereinander kommunizieren. Soll ein Knoten eine Nachricht (`Event`) senden oder empfangen, definiert der Knoten den Namen und den Typ des `Events`. Eine `Route`-Anweisung gibt dann den Weg des `Events` zwischen dem `Event`-Sender und Empfänger an.
Wie in Abbildung 2.1 erkennbar ist, steht die `Route`-Anweisung außerhalb des Szenegraphen, meist am Ende einer VRML-Datei. Sie gibt einen Prozess an, durch den die `Events` propagiert werden, um Änderungen in anderen Knoten vorzunehmen. Ein hervorgerufenes `Event` wird an seine `Route` gesendet und in zeitlicher Reihenfolge anderer `Events` bis zum empfangenen Knoten verarbeitet.
Dieser Prozess kann Felder eines Knoten ändern, weitere `Events` generieren oder die Struktur des Szenegraphen ändern.

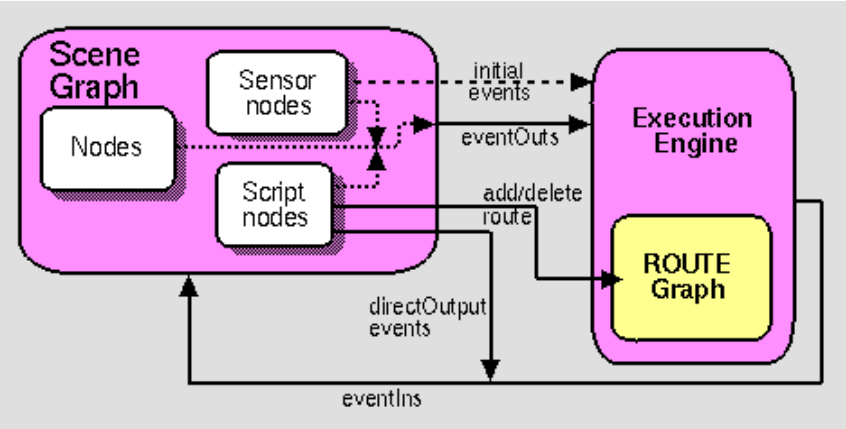



**Abbildung 2.1**: Zusammenhang zwischen Events und Route

b) `Scripts` und `Prototypen`

Der `Script`-Knoten wird genutzt, um die Reaktion der Welt auf eine Veränderung zum Beispiel eine Benutzertätigkeit zu steuern. Einerseits empfangen `Script`-Knoten `Events` von anderen Knoten und können auch Events verschicken, um irgendwo in der Welt Änderungen herbeizuführen. Andererseits können diese Knoten Berechnungen durchführen.
Die Menge der in VRML enthaltenen Knoten kann mithilfe von `Prototypen` durch den Benutzer erweitert werden. `Prototyp`-Definitionen können innerhalb oder außerhalb der VRML-Datei definiert werden. Vom Benutzer definierte `Prototypen` werden mithilfe von `Scripts` gesteuert.
`Trigger` fungieren als Auslöser, um über die `Route` die Welt zu verändern. Diese `Trigger` werden wieder als `Prototypen` definiert.

c) `Sensoren`

`Sensoren` sind die grundlegenden Elemente zur Realisierung von Animationen und benutzerabhängigen Interaktionen in VRML. Der `TimeSensor` generiert `Events` mit zeitlichem Zusammenhang und ist somit die Grundlage für jedes animierte Verhalten. Andere `Sensoren` sind die Grundlage für andere Benutzerinteraktionen und generieren `Events`, je nachdem, wo sich der Benutzer

in der Welt befindet, oder ob er mit der Welt interagiert. So kann zum Beispiel der `TouchSensor` ein `Event` senden, sobald der Benutzer auf eine bestimmte Stelle geklickt hat. `Sensoren` erzeugen lediglich `Events`. Sie müssen über `Route`-Anweisungen mit anderen Knoten kombiniert werden, um eine sichtbare Wirkung in der Welt herbeizuführen.

### 2.1.3.3Nennenswerte, verwendete Knoten

a) `Primitive Shape` und `IndexedFaceSets`

Ein Shape enthält `fields`, die das Aussehen der Wand (`appearance`) und die Geometrie (`geometry`) angeben. In `appearance` wird die Farbe der Wand oder die Textur angegeben. `geometry` hingegen beschreibt die Art der Geometrie zum Beispiel `Box` oder `Sphere`. In einer `geometry` werden unter anderem die Koordinaten der Eckpunkte und die der Textur definiert.

Beispiel 2.1: Einfaches Shape
Ein Haus ist eine Gruppe aus verschiedenen `Shapes`. Eine Wand wird als `Shape` beschrieben, dessen `appearance` die angegebene Textur ist und dessen `geometry` ein `IndexedFaceSet` ist. Im `IndexedFaceSet` werden die Koordinaten der Wand im Feld `coord{point []}` definiert. Das Datenfeld `texCoord{point []}` gibt die Koordinaten mit den Werten von [0,1] an, zwischen denen die Textur ausgeschnitten werden soll. Die Felder `coordIndex [0 1 2 3 -1]` und `texCoordIndex[0 1 2 3 -1]` geben jeweils die Reihenfolge der Punkte an.

Beispiel 2.1: Einfaches Shape

```
DEF Wand Shape {
  appearance Appearance {
    texture ImageTexture {
      url [ "textur.jpg"]
    }
  }
  geometry IndexedFaceSet {
    creaseAngle 0.524
    coord Coordinate {
      point [
        0 0 0
        5 0 0
        5 7 0
        0 7 0
      ]
    }
    coordIndex [ 0 1 2 3 -1 ]
    texCoord TextureCoordinate {
      point [
        0 0
        1 0
        1 1
        0 1
      ]
    }
    texCoordIndex [0 1 2 3 -1]
  }
}
```

### b) Billboard

Ein `Billboard` ist ein Knoten einer Gruppe, der sein Koordinatensystem so verändert, dass sich seine lokale z-Achse immer dem Benutzer zudreht. Da der Knoten eine Gruppe ist, enthält ein `Billboard` weitere Knoten. Falls mehrere `Billboards` definiert werden, dreht sich jedes `Billboard` in seinem eigenen Koordinatensystem. Das Datenfeld `axisOfRotation` gibt bei der Definition die Achse an, um die sich das `Billboard` drehen soll.

### c) Sound

`Sound` repräsentiert eine räumliche Darstellung von Musik in einer VRML-Welt. Um Musik in einer Welt zu definieren, wird zunächst eine Quelle der zu spielenden Musik bereitgestellt. Unterstützte Formate sind zum Beispiel Wave oder Midi. Sollten sich mehrere `Sounds` in einem Bereich der Welt überschneiden und der

Browser mehrere Musikstücke nicht zusammen abspielen können, dann wird der `Sound` mit der höchsten Priorität wiedergegeben. Außerdem kann die voreingestellte Lautstärke des Benutzers vermindert werden.

Der Ursprung (`location`) der Musik ist ein Punkt im Koordinatensystem. Durch zwei Ellipsen und einen Richtungsvektor wird die Umgebung beschrieben, in dem die Musik zu hören ist. Die Ellipsen werden durch `minFront`, `minBack` bzw. `maxFront`, `maxBack` und den Richtungsvektor definiert. Innerhalb der kleineren Ellipse ist die Musik in voller Lautstärke zu hören. Sobald der Benutzer sich nur noch in der äußeren Ellipse befindet, wird die Musik leiser, je weiter er sich vom Ursprung der Musik entfernt. Dabei kann die Musik mit einem Fade-Out ausgeblendet werden. Die Musik wird abrupt abgebrochen, wenn die Ellipsen gleich groß sind.

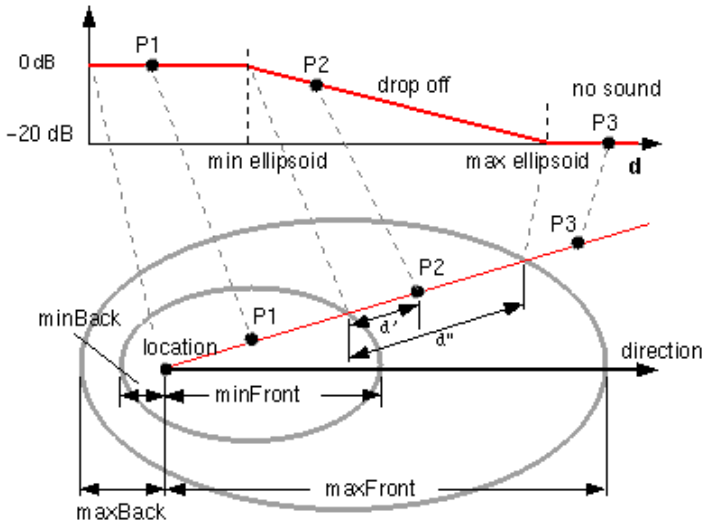



**Abbildung 2.2** : Darstellung des Sounds in VRML

Beispiel:

Im folgenden VRML-Code wird ein `Sound` erstellt, dessen Ursprung die Koordinaten 0,5,0 hat. Das Musikstück „sound.wav“ wird nicht nach dem ersten Abspielen abgebrochen, sondern wiederholt (`loop=true`). Die innere Ellipse kann man sich als Kugel um den Ursprung mit dem Radius 10 vorstellen. Dementsprechend hat die äußere Ellipse den Radius 20. Ein `TimeSensor` wird als Sender der `Events` benötigt, sobald die Musik enden oder beginnen soll. Die

`Route` gibt die Anweisung zum entsprechenden Knoten weiter, der sich dann verändert.

Beispiel 2.2: Sound

```
DEF dad_Sound1 Transform {
  translation 0 5 0
  children [
  DEF Sound1 Sound {
    source DEF audioSound1 AudioClip {
      url [ "sound.wav" ]
      loop TRUE
    }
    minBack 10
    minFront 10
    maxBack 20
    maxFront 20
  }
  ]
}
DEF time TimeSensor {
loop true
}
ROUTE time.cycleTime TO audioSound1.startTime
ROUTE time.cycleTime TO time.stopTime
```

### d) `ProximitySensor` und `Collision`

Ein `ProximitySensor` ist ein Knoten, der `Events` sendet und empfängt, wenn der Benutzer eine bestimmte Region, die als `Box` definiert wird, betritt, verlässt oder sich in ihr bewegt. So kann ein `ProximitySensor` unter anderem, sobald der Benutzer eine Region betritt, ein `Event` senden, um andere Knoten wiederherzustellen.

Ein `Collision`-Knoten dient dazu, dass in ihm enthaltene Knoten vor einem Zusammenprall mit dem Benutzer geschützt werden.

Ein so genannter `HUD` (kurz für: Heads up Display) verbindet beide Arten dieser VRML-Knoten. Ein `HUD` ist ein in einer 3D-Welt feststehendes Objekt. Das heißt, obwohl sich der Benutzer bewegt, steht ein `HUD` immer an der gleichen Stelle des Anzeigefensters, also des Browsers.

Beispiel 2.3: Interaktives `IndexedFaceSet`:

Dieser Beispielcode erzeugt ein `IndexedFaceSet` namens *Fest*, das trotz Bewegungen des Benutzers immer oben links im Browser stehen bleibt, ohne sich mit dem Benutzer zu bewegen.
Der `ProximitySensor` *HudProx* definiert den festen Standort des `IndexedFaceSets`:

```
DEF HudProx ProximitySensor {
  size 500 100 500
  center 0 20 0
}
```

Das `IndexedFaceSet` *Fest* wird mithilfe einer `Collision` erzeugt, sodass ermöglicht wird, dass der Benutzer sich trotz des im Vordergrund stehenden `IndexedFaceSets` frei in der Szene bewegen kann:

```
DEF dad_HUD Transform {
  translation .118 0 12.559
  children [
  DEF HUD Group {
  children [
    DEF dad_collision Transform {
    translation 0 -.25 -.8
    scale .1 .1 .1
    children [
    DEF collision Collision {
      collide FALSE
      children [
      DEF hide_switch_Fest Switch {
        whichChoice -1
        choice [
        DEF dad_Group_Fest_Touch Transform {
          children [
            DEF Group_Fest_Touch Group {
              children [
              DEF Touch_Fest TouchSensor {}
              DEF dad_Fest Transform {
                translation 5.5 5 0
                scale 2 2 .0001
                children [
                DEF Fest Shape {
                  appearance Appearance {
                    material DEF rot Material {
                      ambientIntensity 0.200
                      shininess 0.200
                      diffuseColor 1 0 0
                    }
                  }
                  geometry IndexedFaceSet {
                    creaseAngle 0.524
                    coord Coordinate {
                      point [
                      -.5 .5 .5
```

```
        -.5 -.5 .5
        .5 -.5 .5
        .5 .5 .5
        ]
      }
      coordIndex [
      0 1 2 -1
      0 2 3 -1
      ]
    }}]}]}]}]}]}]}]}]}
```

Gleichzeitig wird ein weiteres interaktives `IndexedFaceSet` namens *Interaktiv* erstellt. Erst wenn dieses `IndexedFaceSet` angeklickt wird, erscheint das `IndexedFaceSet` *Fest* in der oberen linken Ecke.

```
DEF dad_Interaktiv Transform {
  children [
  DEF Interaktiv Shape {
    appearance Appearance {
      material DEF blau Material {
        ambientIntensity 0.200
        shininess 0.200
        diffuseColor 0 0 1
      }
    }
    geometry IndexedFaceSet {
      creaseAngle 0.524
      coord Coordinate {
        point [
        0 0 0
        5 0 0
        5 5 0
        0 5 0
        ]
      }
      coordIndex [
      0 1 2 -1
      0 2 3 -1
      ]
    }
  }
  ]
}
```

Hierzu wird ein `TouchSensor` benötigt, der ein `Event` sendet, sobald das `IndexedFaceSet` *Interaktiv* geklickt wurde.

```
DEF Touch_Interaktiv TouchSensor {}
```

Über die `Route`-Anweisungen wird als erstes das `Event` übermittelt, um das `IndexedFaceSet` *Fest* an den festen Standort zu stellen. Durch die weiteren `Route`-Anweisungen werden die `Events` der `TouchSensoren` weitergeleitet. So kann das Ausblenden oder Erscheinen (engl. hide: ausblenden) festgelegt werden.

```
DEF Xvert0 BooleanTrigger {}
DEF Xvert1 IntegerTrigger{ integerKey -1}
DEF Xvert2 BooleanTrigger {}
DEF Xvert3 IntegerTrigger { integerKey 0}

ROUTE HudProx.position_changed TO dad_HUD.set_translation
ROUTE HudProx.orientation_changed TO dad_HUD.set_rotation

ROUTE Touch_Fest.touchTime TO Xvert0.set_triggerTime
ROUTE Xvert0.triggerTrue TO Xvert1.set_boolean
ROUTE Xvert1.triggerValue TO hide_switch_Fest.whichChoice
ROUTE Touch_Interaktiv.touchTime TO Xvert2.set_triggerTime
ROUTE Xvert2.triggerTrue TO Xvert3.set_boolean
ROUTE Xvert3.triggerValue TO hide_switch_Fest.whichChoice
```

## 2.2 Vizx3D

Vizx3D ist eine Applikation, die es ihrem Anwender ermöglicht, eine 3D-Szene zu erstellen. Es ist das Nachfolgeprogramm des weit verbreiteten Spazz3D (http://www.spazz3d.com).

Dieses Visualisierungswerkzeug ermöglicht dem Anwender sich leicht in der erstellten Szene zurechtzufinden, weil, wie in Abbildung 2.3 erkennbar, zum einen ein ausgereifter VRML-Editor (rechts im Bild) zur Verfügung steht und zum anderen die Szene aus verschiedenen Positionen in 2D angesehen werden kann (links). Vizx3D ist eine Windowsapplikation. Eine Vielzahl grafischer Werkzeuge und „Wizards“ sowie verschiedene Szenenansichten mit Positionierungshilfen erlauben die schnelle Erstellung 3-dimensionaler Szenen.

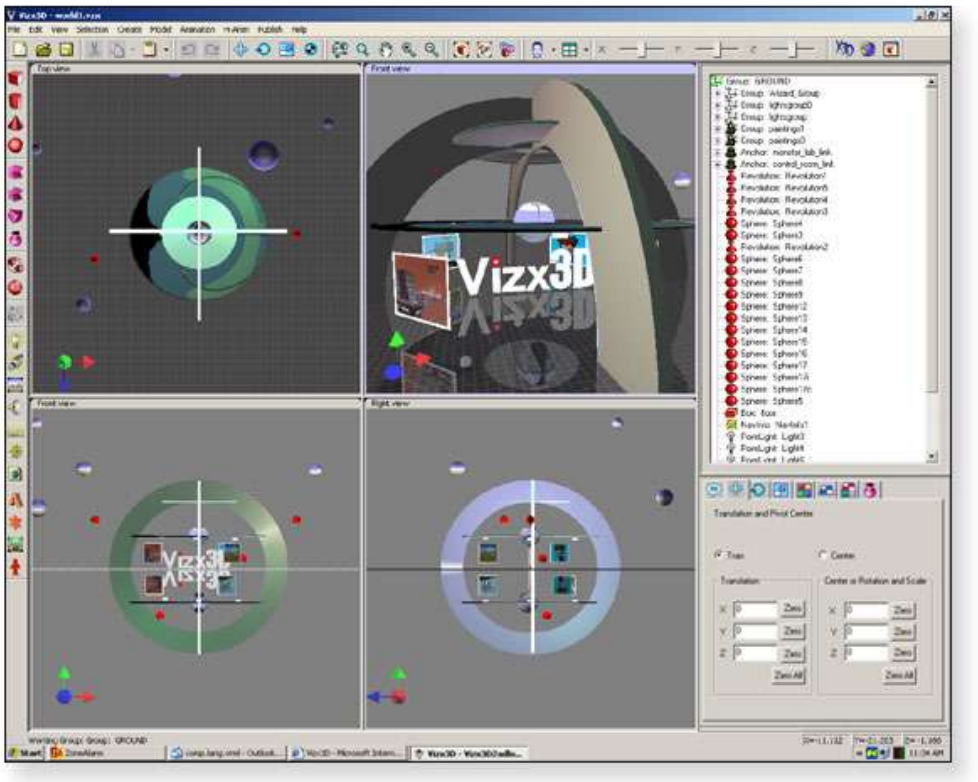



**Abbildung 2.3**: Oberfläche von Vizx3D

Der VRML-Editor gibt den zur Szene gehörenden Szenegraph an. Die im Szenegraph enthaltenen Objekte wie Boxen, Spheres oder einfache IndexedFaceSets können weitestgehend im Editor verändert werden. Nicht nur Translationen, Rotationen und Skalierungen können hinzugefügt oder verändert werden, sondern auch Farben, Texturen und sogar multiple Texturen aus der „Universal Media“, einer eigens für Texturen angelegten Bibliothek.

Nachteilig ist, dass es sehr schwer und umständlich ist, ein einfaches IndexedFaceSet an eine gewünschte Position zu transformieren. Dies wird dadurch hervorgerufen, dass der Benutzer keinerlei Angaben über die direkte Position der Eckpunkte des IndexedFaceSets angeben kann. Die Transformation kann lediglich über translatieren, skalieren oder rotieren um eine bestimmte Achse definiert werden. Sobald um zwei verschiedene Achsen nacheinander rotiert werden soll, steht der Anwender vor einem Problem. Dieses kann zwar gelöst werden, indem vom

Benutzer zuvor komplizierte Rechnungen angestellt werden oder indem das IndexedFaceSet mehrere Elterngruppen besitzt. Diese Elterngruppen führen jeweils eine Rotation aus. Das führt allerdings zu einem weit verzweigten Szenegraph.
Weitere Probleme stellen sich, sobald eine Textur in einer bestimmten Form auf ein IndexedFaceSet gemapt werden soll. Im VRML-Quelltext ist dies einfach durchzuführen, indem die Koordinaten der Textur passend zu den Koordinaten des IndexedFaceSet definiert werden. In Vizx3D ist diese Art der Punktkoordinatendefinition nicht möglich. Also müssen hier Überlegungen angestellt werden, wie eine Textur, mittels der Möglichkeiten von Vizx3D, in die richtige Position gebracht werden kann, ohne eine wirklich perfekte Lösung zu erzielen.

Die 2D-Ansichten lassen keine Wünsche offen. Der Anwender kann nicht nur zwischen den vier Ansichten wechseln oder die Größe verändern, sondern auch die 2D-Szene auch mit der Maus zur gewünschten Ansicht drehen. Per Mausklick kann sich der Anwender die Welt in seinem Standardbrowser anzeigen lassen.
Einfache Modellierungsfunktionen sind ebenfalls integriert. NURBS, „Non Uniform Rational B-Splines“ werden genauso wie polygonale Modelle unterstützt, die sich wie IndexedFaceSets im Detail manipulieren lassen. Das Erstellen von Backgrounds, Viewpoints, Interaktionen und Animationen durch Sensoren scheint für Vizx3D, eine Kleinigkeit zu sein. Für Verhaltensmodellierung erweist sich Vizx3D als Modellierer jedoch als ungenügend, auch Keyframe-Animationen sind nur mühselig zu erstellen. Zwar werden Quelltextdetails für den Nutzer völlig verborgen, dafür ist der Ansatz aber immer noch zu sehr am Szenegraphen orientiert.
Im Gegensatz zu anderen Applikationen bietet Vizx3D einen größeren Komfort und gute Stabilität. Einer der größten Vorteile, die Vizx3D bietet, ist jedoch, dass sich der Anwender mit keinerlei Quelltext auseinandersetzen muss. Er benötigt also keine Kenntnisse in VRML, X3D oder anderen Sprachen.
Wie der Name schon erkennen lässt, können neben VRML97-Szenen auch X3D-Dateien exportiert werden. X3D ist eine Erweiterung, deren Standards zurzeit noch entwickelt werden (http://www.x3dworld.de). Damit gilt diese Anwendung als eine der ersten visuellen X3D-Editoren. Neben Dateiimport von VRML, GIF, JPG und vielen anderen Formaten ist auch ein Export der Szenen als nicht-interaktive AVI-

oder animierte GIF-Dateien oder eine Momentaufnahme der Szene als JPG, GIF oder BMP möglich.
Beim Abspeichern der Welt werden drei Dateien generiert. Zunächst erzeugt Vizx3D eine Datei, mit eigenem Dateiformat und der Endung „vzx“. Des Weiteren wird eine VRML oder X3D-Datei erstellt, in der der eigentliche Quelltext der Szene steht. Außerdem wird eine HTML-Seite generiert, in der die VRML- bzw. X3D-Datei eingebettet wurde.
Alles in allem bietet eine 3D-Entwicklungsplattform große Vorteile, um sich zunächst in eine 3D-Welt einzufinden. Jedoch wird nahezu nie ein optimaler Quelltext erzeugt, der für große und featurereiche Welten notwendig ist.

Für die Zukunft bietet der Unternehmenszusammenschluss von Media Machines und Virtock Technologies für Vizx3D und deren Nachfolgeapplikationen hervorragende Voraussetzungen. Er ist einer der ersten industriellen Schritte, um eine professionelle Lösung für ein Echtzeit-3D-Entwicklungswerkzeug mit X3D, gemäß dem ISO-Standard [ISO-STD], als Grundlage zur Verfügung zu stellen.
Vizx3D ist eine kommerzielle Applikation und in der aktuellen Version 1.2.2 bei der Firma Virtock Technologies unter http://www.vizx3d.com erhältlich.

## 2.3 PHP

PHP, „Hypertext Preprocessor“, ist die grundlegende Skriptsprache, mit der die Applikation erstellt wurde. Sie wurde ursprünglich 1995 von Rasmus Lerdorf entwickelt und ist aktuell in der Version 5 erhältlich, in der viele Verbesserungen gegenüber PHP 4 vorgenommen wurden. Ein herausragendes Merkmal ist die Erweiterung der Sprache um viele Funktionen zur objektorientierten Programmierung.
PHP ist eine für den allgemeinen Gebrauch bestimmte Open Source Skriptsprache. Sie ist besonders gut für die Webprogrammierung geeignet, weil sie in HTML eingebettet werden kann. PHP dient zur Erstellung dynamischer Webseiten.

Veränderliche Informationen werden also bei erneuter Anfrage neu erzeugt. Die Syntax lehnt sich vor allem an C, Java und Perl an.

PHP unterscheidet sich insofern von anderen Programmiersprachen, als dass der Code auf dem Server ausgeführt wird. Das heißt, wenn der Benutzer eine Webseite aufruft, wird der in HTML eingebettete PHP-Quelltext zunächst von einem PHP-Interpreter auf der Serverseite übersetzt und das Ergebnis an den Webserver zurückgesandt. Erst danach erfolgt die Ausgabe an den Client. Auf diese Art kann der Benutzer den zugrunde liegenden Quelltext nicht mehr erkennen. Die Ausgabe beschränkt sich jedoch nicht nur auf HTML. Mit PHP können ebenfalls Bilder, PDF-Dateien oder Flash-Animationen dynamisch generiert werden. Weiterhin kann PHP Dateien automatisch erzeugen und im Dateisystem speichern. Diese Funktionalität wird auch in der Applikation mehrfach angewandt, um VRML-Dateien zu erstellen, anzuzeigen und abzuspeichern.

Ein Vorteil der serverseitigen Ausführung ist, dass beim Client (Browser) keine speziellen Fähigkeiten erforderlich sind oder Inkompatibilitätsprobleme auftreten können, wie es zum Beispiel bei Javascript und verschiedenen Browsern der Fall ist. Somit ist PHP plattformunabhängig, unterstützt also die meisten der heute gebräuchlichen Webserver und kann auf allen gängigen Betriebssystemen verwendet werden.
Dies ist auch für andere Ressourcen von großer Bedeutung, zum Beispiel brauchen Datenbanken keine direkte Verbindung zum Client.
Von Nachteil ist, dass jede Aktion des Benutzers erst bei einem erneuten Aufruf der Seite bearbeitet werden kann.
Obwohl der Umfang von PHP enorm ist und für professionelle Programmierer ausreichend Funktionen bietet, ist PHP dennoch leicht zu erlernen, da Einsteiger einen schnellen Zugang zu den Funktionen und deren Handhabung finden.
Hervorzuheben ist, dass PHP eine breite Datenbankunterstützung, sowie zahlreiche, zusätzliche Funktionsbibliotheken anbietet. Sogar Komprimierungswerkzeuge wie gzip oder bz2 sind vorhanden.

Die Softwarelizenz „PHP License“, unter der PHP zur Verfügung gestellt wird, erlaubt die freie Verwendung und Veränderung der Quelltexte. Die Software zu PHP kann kostenlos aus dem Internet unter http://de3.php.net/downloads.php bezogen werden.

Es folgt ein Beispiel, das zeigt, wie PHP in HTML eingebettet werden kann. Hierbei kann beliebig zwischen PHP- und HTML-Quelltext gewechselt werden.
Beispiel 2.4: PHP eingebettet in HTML:

```
<?php
  //PHP-Quelltext
  $halloWelt = "Welt";
?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Bachelorarbeit von Katrin T&ouml;pler</title>
</head>
<body>
<!--Hier wird PHP in HTML eingefügt:-->
<?php
  //Ausgabe der Variablen $halloWelt:
  echo "Hallo ".$halloWelt."!";
?>
</body>
</html>
```

Dieser Quelltext gibt eine HTML Seite mit dem Text „Hallo Welt!” aus.

# II. Realisierung der Applikation

# 3 Vorhandene Daten

Eine Stadt als virtuelle 3D-Welt darzustellen, wurde von der Stadt Osnabrück in Auftrag gegeben. Die Aufgabe bestand zunächst darin, eine virtuelle Welt von einem Teil der Osnabrücker Altstadt anzufertigen. Hierzu wurden mir folgende Daten in Form von Grafiken von der Stadt Osnabrück zur Verfügung gestellt.

Ein Stadtplan mit den Grundrissen der Häuser dient als Grundlage für die 3D-Szene:

**Abbildung 3.1**: Grundriss der Osnabrücker Altstadt

Die gelben Flächen stellen einfache Gebäude dar. In der Osnabrücker Altstadt grenzt fast immer ein Haus an das andere. In der Abbildung lässt sich leicht erkennen, dass nahezu keines der Häuser einen symmetrischen Grundriss hat. Das liegt vor allem daran, dass die Häuser in diesem Teil der Stadt schon einige Jahrhunderte alt sind. Die rot markierten Flächen bezeichnen öffentliche Gebäude, oben links das Rathaus, oben rechts die Marienkirche und unten die Stadtbibliothek. Diese Flächen fallen nicht wegen ihres ungewöhnlichen Umrisses aus dem Schema eines normalen Hauses heraus, sondern weil sie seltene Gebäudeformen haben.

Diese Grafik gibt also die Eckpunkte der Häuser, die Breiten und die Tiefen im maßstabsgetreuen Verhältnis an.
Die folgenden Abbildungen dienen zur Ermittlung der Haushöhen.

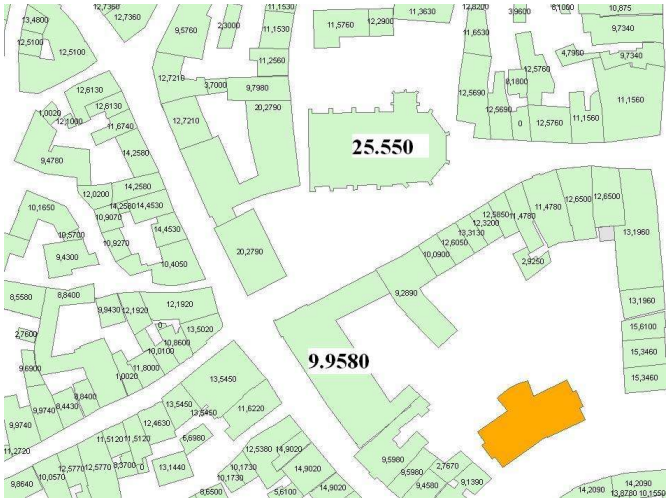



**Abbildung 3.2**: Minimale Höhe der Gebäude


**Abbildung 3.3**: Maximale Höhe der Gebäude

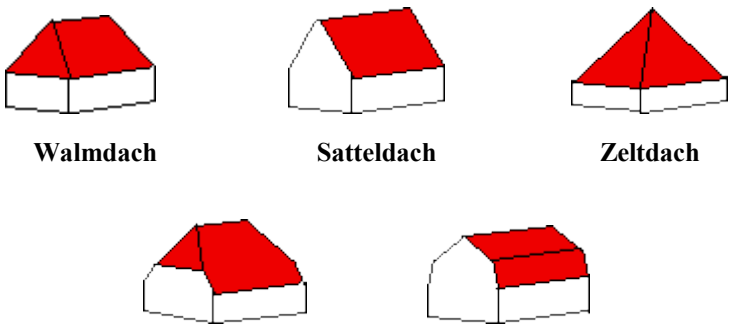
Die Abbildungen 3.2 und 3.3 stellen von der Stadt erhobene Daten der Häuserhöhen dar. Die Daten wurden erfasst, indem die Häuser von einem Flugzeug aus eingescannt und die Höhen berechnet wurden. Es wurden pro Haus zwei Höhen berechnet, die

minimale und die maximale Höhe des Hauses. Aufgrund der Ungenauigkeit dieser Werte sind sie für eine maßstabsgetreue Abbildung nicht verwertbar und nur bedingt zur Beurteilung von Höhendifferenzen anwendbar.

Abbildung 3.4 zeigt die von oben abfotografierten Häuser und dient zur möglichst genauen Darstellung der Hausdächer.

**Abbildung 3.4**: Hausdächer um den Rathausplatz

Um die Häuserformen näher zu betrachten und dahingehend zu untersuchen, welche Formen in der Altstadt vorhanden sind, habe ich die betreffenden Häuser fotografiert. Bei näherer Betrachtung der Häuser stellt man fest, dass folgende Hausformen in der Altstadt vorhanden sind:



Diese zur Verfügung gestellten Daten sollen so verarbeitet werden, dass eine komplette Welt entsteht. Wobei festgestellt werden muss, für welche Objektart welche Angaben notwendig sind und welche Angaben vom Benutzer am einfachsten zu ermitteln sind. So ist es nicht sinnvoll, den Benutzer nach dem Dachwinkel zu fragen, um die Form eines Walmdachs zu erhalten, da ein Dachwinkel schwer errechenbar oder messbar ist. Außerdem bietet es sich nicht an, nach den Breiten der einzelnen Hausetagen zu fragen, da hierbei zu viele Einschränkungen entstehen, um ebene Dachseiten des Hauses zu garantieren. Um dem Anwender die Benutzung also zu erleichtern, werden die einzelnen Höhen der Hausetagen bestimmt. So kann sichergestellt werden, dass ein Krüppelwalmdach möglichst nach Vorstellungen des Benutzers erbaut wird, da Einschränkungen bezüglich der Eingabedaten fast völlig außer Acht gelassen werden können.

# 4 Ermittlung der Eingabedaten

In diesem Kapitel wird erläutert, wie die grundlegenden Informationen zur Dateneingabe ermittelt werden können. Die Informationen, die benötigt werden, um ein Objekt zu definieren, sind von Objekt zu Objekt verschieden und differenzieren sich in der Art der Beschaffenheit jedes einzelnen Objekts.

## 4.1 Grundkoordinaten

Um Häuser an die richtige Position in der 3-dimensionalen Welt zu stellen, ist es zwingend notwendig, deren Grundkoordinaten zu kennen. Wenn ein Stadtplan mit den Grundrissen der Häuser zur Verfügung steht, ist die Ermittlung der Grundkoordinaten leicht.

Wird der Stadtplan in einem Bildbearbeitungsprogramm angezeigt, kann sich der Benutzer in nahezu jedem Editor, durch Bewegen der Maus über die Grafik, die Pixelkoordinaten der aktuellen Mausposition anzeigen lassen. Allerdings müssen die Pixelkoordinaten des Bildbearbeitungsprogramms, wie folgt, umgerechnet werden, sodass sie die Grundfläche eines 3-dimensionalen Objektes ergeben.

In einer VRML-Welt stellt die x,z-Ebene die Grundfläche dar. Die Höhe der Welt wird auf y-Achse angegeben (vgl. Abbildung 4.1).

Das Koordinatensystem der Grafik und das der 3D-Welt passen nicht zusammen. In der 3D-Szene sollte jedoch das Haus in der Mitte der Grafik im Ursprung der x, z-Ebene liegen, damit die Mitte der Grafik die Mitte der Welt ist. Also müssen die Pixelkoordinaten der Grafik so umgerechnet werden, dass der Nullpunkt in der Welt dem Nullpunkt der Pixelkoordinaten in der Grafik entspricht. Die Achsen des Koordinatensystems der Grafik läuft von oben links in positiver x- bzw. y-Richtung nach rechts bzw. unten (vgl. Abbildung 4.2).

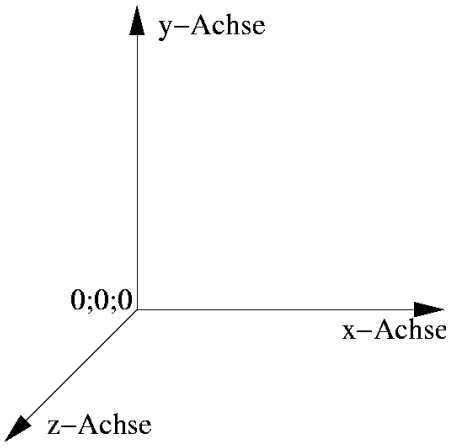



**Abbildung 4.1**: Koordinatensystem einer 3D-Szene

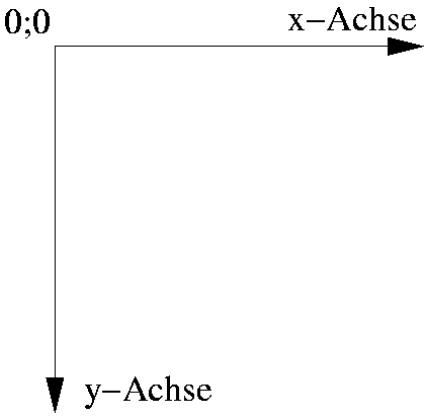



**Abbildung 4.2**: Koordinatensystem eines Bearbeitungsprogramms

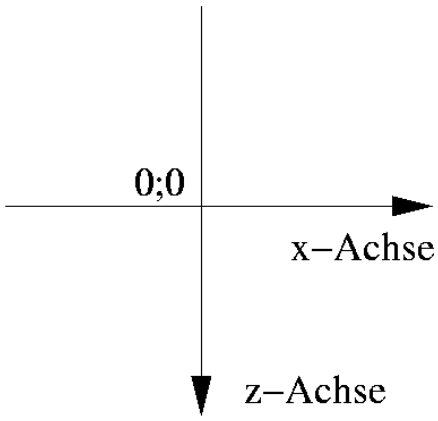



**Abbildung 4.3**: Grundfläche des 3D-Koordinatensystems

Für die Umrechnung gilt also:

$$(x_{welt}, z_{Welt})^T = \left( x_{Grafik} - \frac{x_{\max,Grafik}}{2}, y_{Grafik} - \frac{y_{\max,Grafik}}{2} \right)^T$$

Die Werte $x_{max,Grafik}$ und $y_{max,Grafik}$ geben jeweils die Größe der Grafik an. $X_{Grafik}$ und $y_{Grafik}$ stehen für die Position des Mauszeigers und $x_{Welt}$ und $z_{Welt}$ bezeichnen die neuen Koordinaten in der 3D-Szene.
Sobald die zu bebauende Fläche in der 3D-Welt auf die Pixelgröße der Grafik skaliert wurde, stimmen die umgerechneten Pixelkoordinaten mit den Grundkoordinaten überein.

Um ein Haus darstellen zu können sind Angaben zu drei Grundkoordinaten A, B und C erforderlich. Punkt A und B definieren die unteren Eckpunkte der Hausfront. Sie geben sowohl die Position des Hauses, als auch dessen Breite an. Punkt C definiert zusammen mit Punkt A die Haustiefe. Die vierte Grundkoordinate lässt sich aus den Vektoren der ersten drei berechnen. Weitere Berechnungen hierzu werden in den Abschnitten 5.3.2 und 5.3.3. näher erläutert.

## 4.2 Höhen

Da für die Höhen der Häuser keine maßstabsgetreuen Daten vorliegen, ist es nicht möglich die längentreuen Höhen zu ermitteln.
Eine Möglichkeit die Höhen annähernd festzulegen ist, die Bilder der Häuser zu betrachten. Mit der Grafik lässt sich näherungsweise das Verhältnis zwischen Breite und Höhe bestimmen. Die genaue Hausbreite kann mithilfe der Pixelkoordinaten des Grundrisses ermittelt werden. Mit der genauen Breite und dem Verhältnis von Hausbreite zu Haushöhe kann somit auch die Höhe näherungsweise bestimmt werden.

## 4.3 Dachkantenlängen und andere nicht ermittelbare Daten

Die Höhen zu errechnen, ist zwar umständlich, aber möglich. Allerdings gibt es Daten, die nur schätzbar sind, zum Beispiel liegen für die Länge der Dachkanten keine Daten vor. Außerdem ist es nicht möglich, andere Objekte, wie Menschen, Bäume oder Sträucher maßstabsgetreu anzulegen.

## 4.4 Texturen

Als Texturen stehen die Texturen der Universal Media-Bibliothek von Vizx3D zur Verfügung. Diese Bibliothek stellt einige Texturen, zum Beispiel für Hintergründe, Dächer, Steine etc. bereit. Ansonsten dienen die Bilder der Häuser als realitätsnahe Texturen.

Texturen müssen vom Benutzer nicht zuvor in die richtige Größe skaliert werden. Er sollte die Textur jedoch auf die passende Größe zuschneiden. Die Applikation behandelt Texturen auf unterschiedliche Weise.
So wird bei einem Haus die untere Breite der Textur vollständig auf das `IndexedFaceSet` gemapt, wobei die Höhe der Textur an die Höhe des `IndexedFaceSets` angepasst wird (vgl. Abbildung 4.4).

**Abbildung 4.4:** Viereckiges IndexedFaceSet der Größe 5x5

Sollte das `IndexedFaceSet` ein Dreieck darstellen wie zum Beispiel am Hausgiebel, so wird die Textur wie oben beschrieben skaliert und an den Seiten passend abgeschnitten (vgl. Abbildung 4.5).

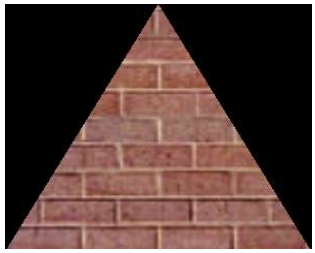
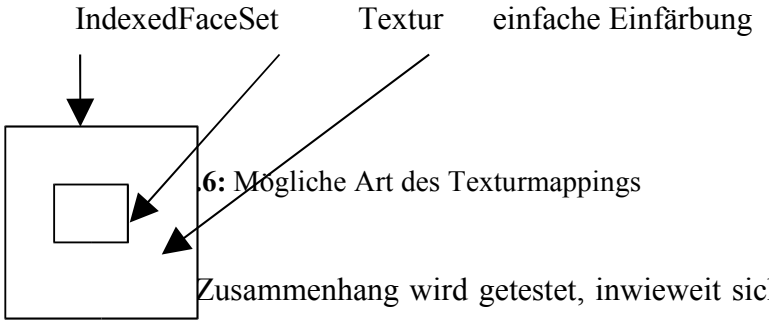


**Abbildung 4.5:**Dreieckiges IndexedFaceSet der Größe 5x4

Möchte der Anwender die Textur auf eine andere Art auf das `IndexedFaceSet` legen, kann er dafür eine Wand erstellen. Eine allein stehende Wand besteht aus einem IndexedFaceSet und einer Textur. Hier kann der Anwender selbst bestimmen, wie die Textur auf das `IndexedFaceSet` gelegt wird indem er die Koordinaten des `texCoord`-Feldes selbst bestimmen kann.

Eine Textur kann selbstverständlich auf mehrere `IndexedFaceSets` gemapt werden. Hierbei treten für den Anwender jedoch Probleme auf. Die Texturen eines Daches werden zum Beispiel immer nach Breite und Höhe des `IndexedFaceSets` skaliert. Wird nun auf dem Giebel und auf der Dachseite eines Hauses die gleiche Textur verwendet, so kann es vorkommen, dass die Textur sehr unterschiedlich skaliert wird. Dieser Fall tritt aber nur dann ein, wenn die Größen der `IndexedFaceSets` sehr unterschiedlich sind.

Um Speicherplatz zu sparen, gab es die Idee, nur dort Texturen auf die `IndexedFaceSets` zu mappen, wo sie auch wirklich notwendig sind, also wo sich zum Beispiel Wände eines Hauses voneinander unterscheiden. Der Rest des `IndexedFaceSets` wird in einer Farbe getönt. Eventuell könnte so der Speicherbedarf auf dem Server durch kleinere VRML-Dateien und Texturen vermindert werden.




**6:** Mögliche Art des Texturmappings

Zusammenhang wird getestet, inwieweit sich diese Art des Mappings auf die Größe der VRML-Datei auswirkt. Es werden `IndexedFaceSets` erstellt, die sich lediglich in der Art des Mappings unterscheiden. Nun werden zugehörige VRML-Dateien angelegt, in die die angelegten `IndexedFaceSets` mehrere Male kopiert werden. Das Ergebnis ist, dass die VRML-Datei, in der die Textur vollständig auf das `IndexedFaceSet` gelegt wird, kleiner ist, als die andere. Denn der VRML-Interpreter muss weder die Lage der Textur auf dem `IndexedFaceSet`, noch die Farbe des `IndexedFaceSets` berechnen.

# 5 Verarbeitung der angegebenen Daten

Die Applikation bietet an, verschiedenartige Objekte herzustellen. Abgesehen von den schon erläuterten Hausformen können folgende Objekte angelegt werden.

- Wand:
  Um zum Beispiel die Welt zu begrenzen, kann der Benutzer eine allein stehende Wand anlegen. Diese einzelne Wand ist wie eine Wand eines Hauses auch als IndexedFaceSet definiert. Somit kann der Benutzer ihren Standort, Größe und Textur definieren.

- Bäume, Personen, Sträucher etc.:
  In die 3D-Welt kann der Benutzer Personen, Bäume oder ähnliches in Form eines Billboards anlegen. Die Position, Größe und Textur kann der Benutzer selbst definieren.

- Hintergrund:
  Der Hintergrund der Applikation enthält nicht nur die Texturen der Flächen, die die Welt umgeben, außerdem werden Standpunkte des Betrachters, die Art und Position des Lichtes und die Größe und die Texturen der Bodenfläche definiert.

- Unterlegung mit Musik:
  Der Benutzer hat die Möglichkeit seine Welt mit Musik zu unterlegen. Er kann mehrere Musikquellen in einer VRML-Welt angeben, die an unterschiedlichen Regionen in der 3D-Szene zu hören sind. Weiterhin kann er die Musik auf verschiedene Arten ausblenden.

- Interaktives Textfeld:
  Eine besondere Eigenschaft der Applikation ist, dass dem Benutzer die Möglichkeit gegeben wird, nicht nur eine multimediale, sondern auch eine interaktive Welt zu erstellen. Der Anwender kann ein interaktives Textfeld hinzufügen. Dieses Textfeld erscheint, sobald auf bestimmte vom Benutzer festgelegte Objekte geklickt wird. Wenn der Benutzer auf das Textfeld klickt, verschwindet es wieder. Das Textfeld bleibt in der oberen Ecke des Browsers stehen, auch wenn der Anwender weiter durch die Welt geht. So hat der Anwender der Applikation die Möglichkeit Informationen in die 3D-Szene einzubauen.

Der Anwender hat die Möglichkeit, alle Objektarten in eine gemeinsame Welt einzufügen. So kann eine Welt aus vielen einzelnen Objekten bestehen. Eine Welt kann außerdem auch andere Welten enthalten.

## 5.1 Anlegen eines Benutzerverzeichnisses

Wenn sich ein Benutzer zum ersten Mal in die Applikation einloggt bzw. registriert, werden seine Benutzerdaten abgespeichert und ein neuer, eindeutiger Ordner auf dem Server für ihn angelegt. In diesen Ordner werden die Daten der von ihm angelegten Objekte und deren VRML-Dateien abgespeichert. Die Unterteilung in verschiedene Ordner dient dazu, dem Anwender Sicherheit, Zuverlässigkeit und Benutzbarkeit zu garantieren.
Sobald der Benutzer sich unter dem entsprechenden Benutzernamen mit Passwort einloggt, wird mit PHP eine `Session` gestartet, bei der die Konstante „USERVERZEICHNIS“ in den dem Benutzer zugewiesenen Ordner umbenannt wird. Die folgende Abbildung zeigt die Startseite, die nach einem erfolgreichen Einloggen erscheint.


**Abbildung 5.1:** Startseite der Applikation

Je nach Objektart werden vom Anwender verschiedene Angaben verlangt, um ein Objekt zu erzeugen. Somit werden dem Anwender verschiedene Eingabeformulare für die Objektarten angezeigt. Abbildung 5.2 zeigt ein Eingabeformular für ein Walmdach. Nachdem der Benutzer die erforderlichen Daten angegeben hat, werden seine Angaben verarbeitet. Gegebenenfalls wird schon vorab eine VRML-Datei angezeigt, sodass der Benutzer direkt die Möglichkeit hat, die Visualisierung seiner Angaben zu betrachten (siehe Abbildung 5.2). Die Angaben werden jeweils zu einer VRML-Datei verarbeitet und in eine einfache Textdatei geschrieben. In den folgenden Kapiteln wird genauer darauf eingegangen, wie diese Dateien erzeugt und abgerufen werden.

Nachdem der Benutzer erforderliche Daten angegeben hat, werden seine Angaben verarbeitet. Gegebenenfalls wird schon vorab eine VRML-Datei angezeigt, sodass der Benutzer direkt die Möglichkeit hat, die Visualisierung seiner Angaben zu betrachten (siehe Abbildung 5.3). Die Angaben werden jeweils zu einer VRML-Datei verarbeitet

und in eine einfache Textdatei geschrieben. In den folgenden Kapiteln wird genauer darauf eingegangen, wie diese Dateien erzeugt und abgerufen werden.

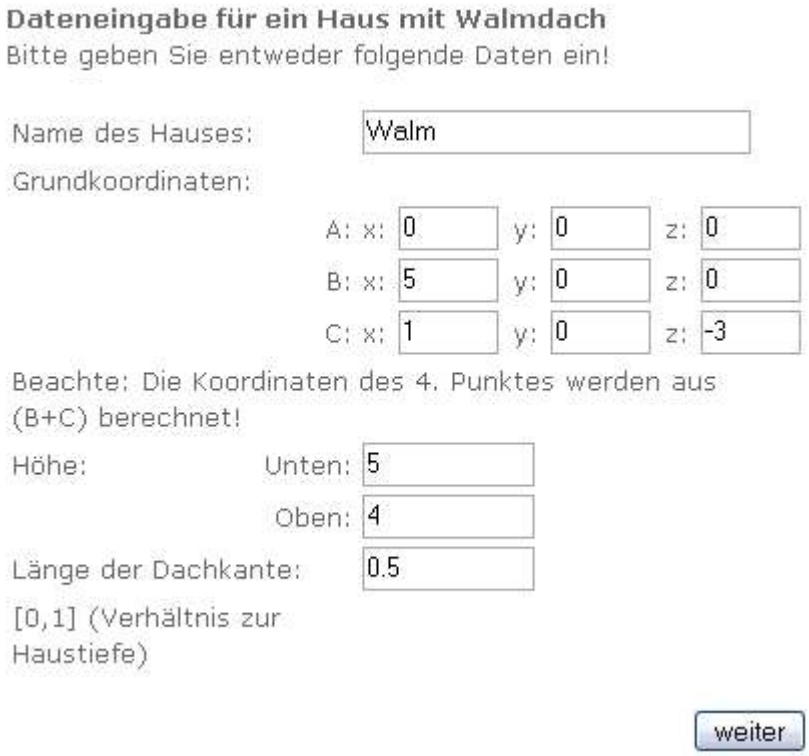



**Abbildung 5.2**: Dateneingabe für ein Haus


**Abbildung 5.3**: Formular zur Wandeingabe eines Haus

## 5.2 Sinn und Aufbau einer Textdatei

Die Aufgabenstellung ist zu komplex, um eine Datenbank anzubinden. Aus diesem Grund werden Textdateien benutzt, um Informationen und Angaben über die Objekte abzuspeichern. Auf diese Art dienen die Textdateien dazu, auch im nachhinein Objekte ändern zu können.

Nachdem alle Angaben für ein Objekt von einem Benutzer bestimmt wurden, wird die Textdatei mithilfe von PHP erzeugt. Der Aufbau einer solchen Datei ist im Grunde immer gleich. Am Anfang des Dokuments werden die Objektart und der Objektname aufgelistet, um später diese beiden Informationen möglichst schnell aus dem Dokument lesen zu können. Danach folgen die restlichen Informationen in der Form, dass in der Zeile unter dem Parameter jeweils der Wert steht. Diese Art des Schreibens bzw. Speicherns der Werte ist von Vorteil, da PHP gegenwärtig leider noch keine Möglichkeit anbietet, den Inhalt von Dateien zu verändern. Sobald eine Datei in PHP geöffnet wird, wird der Dateiinhalt gelöscht. Um einen Teil des Dateiinhalts zu ändern, muss also zunächst die Datei in ein Array umgewandelt werden. Daraufhin werden die Werte des Arrays geändert. Erst danach wird das Array in die Datei mit dem ursprünglichen Dateinamen geschrieben. Da die Umwandlung einer Datei in ein Array zeilenweise geschieht, ist die Form, die Informationen zeilenweise in die Textdatei zu schreiben, am sinnvollsten.
Je eine Schreib- und eine Lesemethode werden für eine Objektart erzeugt. Dies ist notwendig, weil die Anzahl der Daten für jede Objektart unterschiedlich ist. Da PHP das Überladen von Methoden nicht unterstützt, müssen für jede Objektart eigens benannte Schreib- und Lesemethoden geschrieben werden. In den Lesemethoden werden die Daten aus den Textdateien gelesen und in ein Array, das zurückgegeben wird, geschrieben.

## 5.3 Aufbau der VRML-Dateien

Eine VRML-Datei zu erstellen, ist aufwändiger als die Erstellung einer Textdatei. Wie oben erwähnt, bietet PHP keine direkte Möglichkeit, den Inhalt einer Datei zu ändern. Dem Anwender der Applikation wird es aber meist zur Verfügung gestellt, die vorläufigen VRML-Dateien, noch vor Eingabe aller notwendigen Daten soweit wie möglich anzusehen und somit direkt die Veränderungen in der Welt durch getätigte Eingaben beobachten zu können. Um dieses Feature zu gewährleisten, wird die anzuzeigende VRML-Datei nach der Eingabe verändert.

Bei der Implementation wird ausgenutzt, dass VRML als Grundstruktur einen Szenegraph besitzt. So stellt zunächst das Anlegen einfacher Objekte, wie einer Wand, eines `Billborads` oder auch eines Hauses kein Problem dar. Dies wird im nächsten Abschnitt näher erklärt.

### 5.3.1 Herkömmliche Objekte

Herkömmliche Objekte wie ein Haus, das aus `IndexedFaceSets` besteht, oder ein `Billboard` können im Rumpf eines VRML-Quelltextes nacheinander aufgelistet werden. Es existieren wieder für jede Objektart eine oder mehrere Schreibmethoden.

In einer Schreibmethode werden als erstes die richtigen Grundkoordinaten berechnet. Aus ihnen lassen sich die Hausdrehung bzw. der richtige Standort in Bezug auf den richtigen Winkel zum Ursprung und auch die Positionen der einzelnen `IndexedFaceSets` ermitteln. Im folgenden Kapitel wird genauer darauf eingegangen, wie diese Werte erzielt werden. Funktionen werden aufgerufen, die den Quelltext für einen bestimmten Teil der VRML-Welt zurückliefern. Bei einem Haus wird zum Beispiel erst dann der Text für die einzelnen Wände geschrieben, nachdem der Text für den Standort des Hauses ausgegeben wurde. So kann leicht die Struktur eines Hauses oder anderer einfacher Objekte im Szenegraph beschrieben werden.

### 5.3.2 Berechnungen des Grundbaus eines Hauses

Da dem Anwender die Dateneingabe erleichtert werden soll, werden von ihm zur Erstellung eines Hauses lediglich drei Grundkoordinaten verlangt. Aus diesen Grundkoordinaten lässt sich sowohl die Breite bzw. Tiefe des Hauses, als auch die Stellung des Hauses in der Szene ermitteln. Die Tiefe des Hauses ist für die spätere Berechnung des Daches und der Dachkantenlänge von Bedeutung. Die Stellung des Hauses und damit der `IndexedFaceSets` in der 3D-Welt lässt sich wie folgt berechnen.

Um die Rechnungen zu vereinfachen, werden die `IndexedFaceSets`, aus denen ein Haus besteht, so definiert, als stünde das Haus mit dem Punkt A im Ursprung und dem Punkt B auf der positiven x-Achse (vgl. Abbildung 5.4).

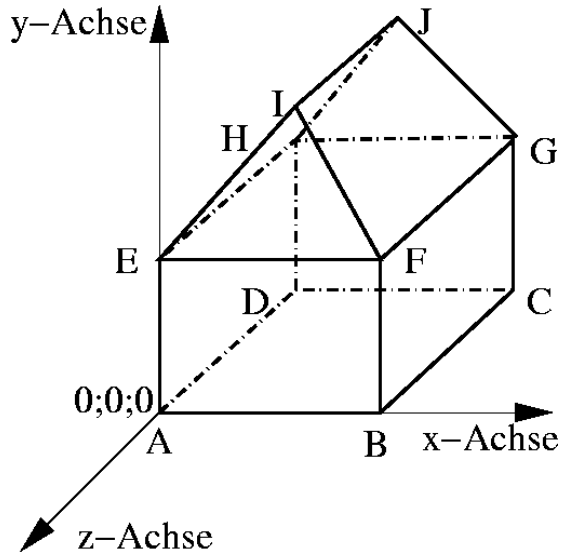



**Abbildung 5.4**: Beschriftung der Hausecken

Erst später wird das Haus als ganzes Objekt um den Winkel a um die y-Achse rotiert und dann um den Vektor v translatiert (vgl. Abbildung 5.5).

Zunächst scheint dieses Vorgehen umständlich. Jedoch werden so die Berechnungen für die `IndexedFaceSets` vereinfacht und ein optimaler Quelltext der VRML-Welt erzielt.

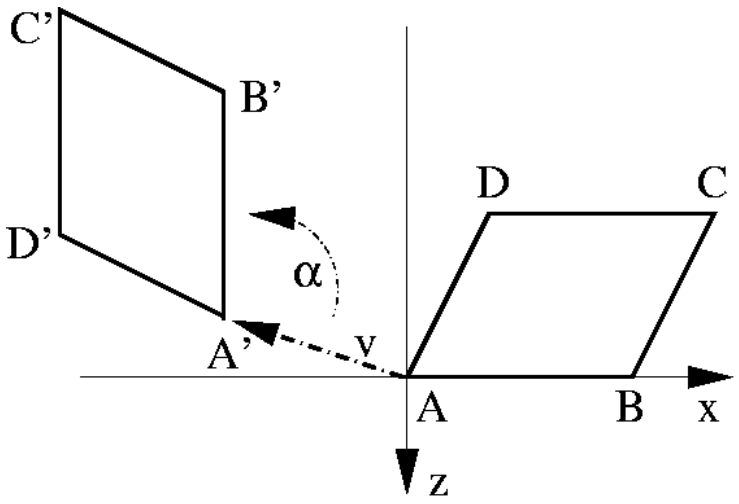
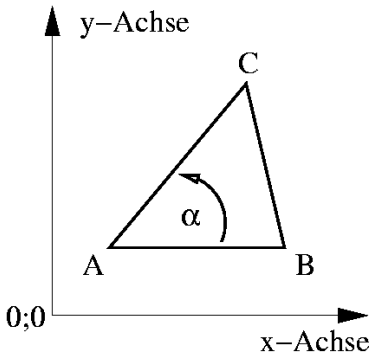



**Abbildung 5.5**: Rotation und Translation eines Hauses

Um also das Haus an die richtige Position in der VRML-Welt erbauen zu können, muss der Winkel der Rotation und der Vektor der Translation berechnet werden.
Zur Berechnung des Vektors ist nicht viel Aufwand notwendig. Es wird festgelegt, dass das Haus mit dem Punkt A im Ursprung steht. Demnach stellt der Vektor vom Ursprung zum Punkt A den Translationsvektor dar.
Mühsamer ist es die richtige Berechnung des Rotationswinkels zu erlangen. Für einen Winkel α in einem beliebigen Dreieck A, B, C gilt:

$$\alpha = \frac{a \otimes b}{|a| \bullet |b|}$$

Skizze:



Auf den ersten Blick lässt sich keine Schwierigkeit erkennen. Ein IndexedFaceSet hat aber die Eigenschaft, dass es nur eine Vorderseite hat, von der hinteren Seite ist es unsichtbar. Außerdem ist zu beachten, dass das Objekt wie eine Wand von zwei Seiten betrachtet werden kann, obwohl dieselben Koordinaten der Punkte A und B angegeben wurden (vgl. Abbildung 5.6).

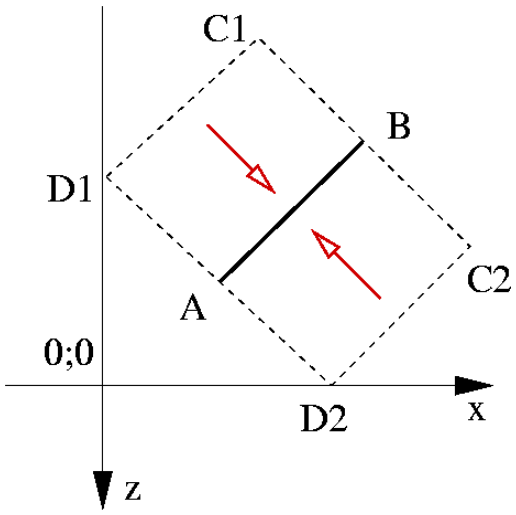



**Abbildung 5.6**: Mögliche Vorderseiten eines IndexedFaceSets

Es ist ersichtlich, dass für die Berechnung des Winkels zwei Arten und somit auch zwei Formeln vorliegen:

$$\alpha = \frac{a \otimes b}{|a| \bullet |b|} \quad \text{und} \quad \alpha' = 180° + \alpha$$

Wann welche Formel gilt, wann $\alpha$ also größer oder kleiner als 180° ist, soll Abbildung 5.7 veranschaulichen. In dieser Abbildung werden die verschiedenen Hausstellungen dargestellt. Die langen Strecken zeigen das von oben betrachtete `IndexedFaceSet`. Die kleinen Strecken vor den `IndexedFaceSets` geben die Ansichtsseite oder Vorderseite des `IndexedFaceSets` an. Die rot dargestellten Strecken zeigen das ursprüngliche, noch nicht rotierte, `IndexedFaceSet`.

Aus Abbildung 5.7 ist leicht ersichtlich, dass $\alpha \geq 180°$ ist, wenn $z_A \leq z_B$ gilt.

Somit kann durch eine zusätzliche if-Abfrage eine umständliche Umrechnung des Winkels ersetzt werden.

Die anderen Eckpunkte der jeweiligen `IndexedFaceSets` lassen sich aus den Grundkoordinaten und der Höhe des Hauses berechnen.

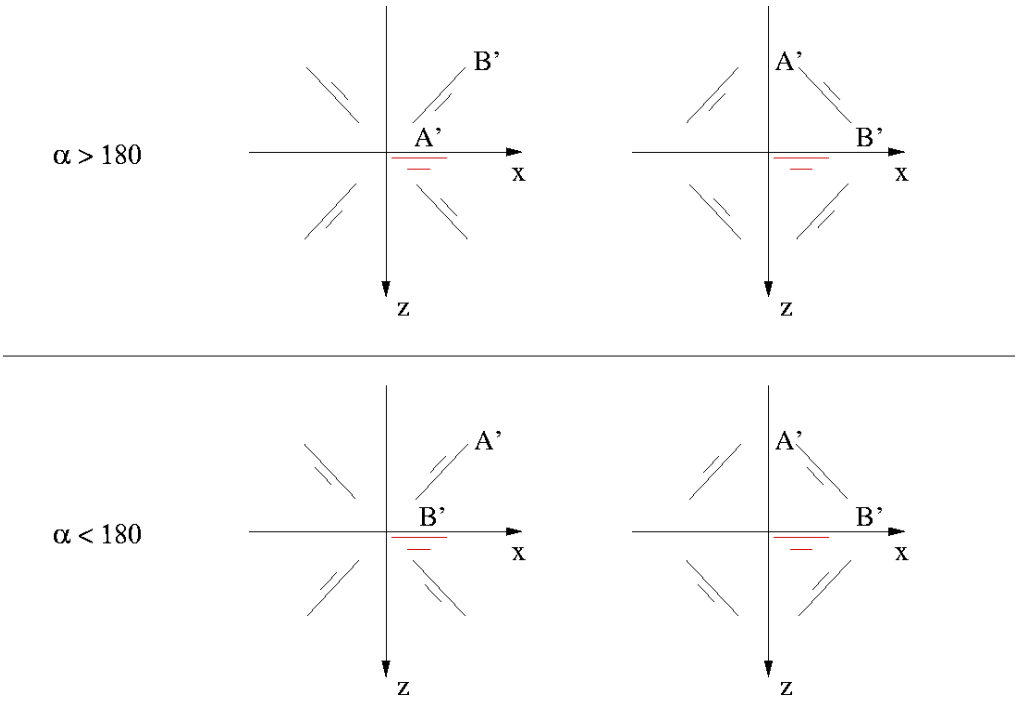



**Abbildung 5.7**: Hausstellung bzgl. des Rotationswinkels

### 5.3.3 Berechungen eines Daches am Beispiel eines Walmdachs

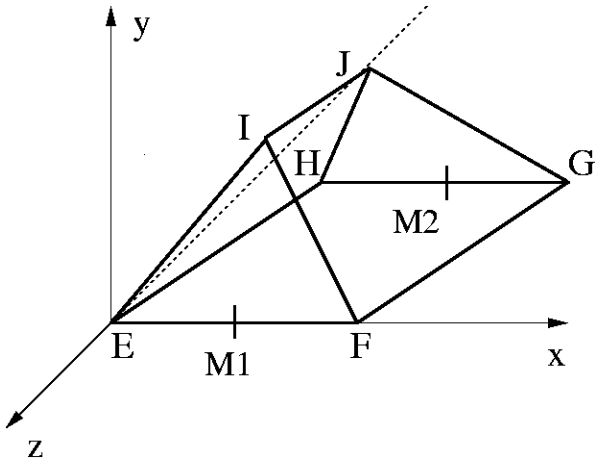


**Abbildung 5.8**: Beschriftung der Dachpunkte in 3D

Ein Dach eines Hauses hat folgende Eigenschaften:

- Es existiert eine Dachhöhe.
- Die Dachkante ist die Strecke IJ.
- Die Dachkantenlänge ist abhängig von der Haustiefe.
- Die Punkte M1, M2 sind Mittelpunkte der Strecken EF bzw. GH.
- Die Punkte M1, M2, J und I liegen in einer Ebene.
- Die Dachkante IJ liegt parallel zu den Strecken EH und FG, somit ergibt sich ein Parallelogramm als Grundfläche (siehe Abbildung 5.9).

Um ein Dach darstellen zu können, werden die Eckpunkte der einzelnen Dachseiten benötigt. Die vier unteren Punkte E, F, G, H sind leicht ermittelbar. Sie ergeben sich aus den Grundkoordinaten des Hauses und der dazu addierten unteren Haushöhe. Die Dachkantenlänge berechnet sich aus dem angegebenen Verhältnis zur Haustiefe. Gibt der Anwender zum Beispiel als Verhältnis den Wert 1 an, so wird ein Satteldach erzeugt. Im Gegenzug dazu wird beim Wert 0 ein Zeltdach erstellt.

Um nun die Punkte I und J zu ermitteln, werden zunächst die Punkte M1 und M2 ermittelt.

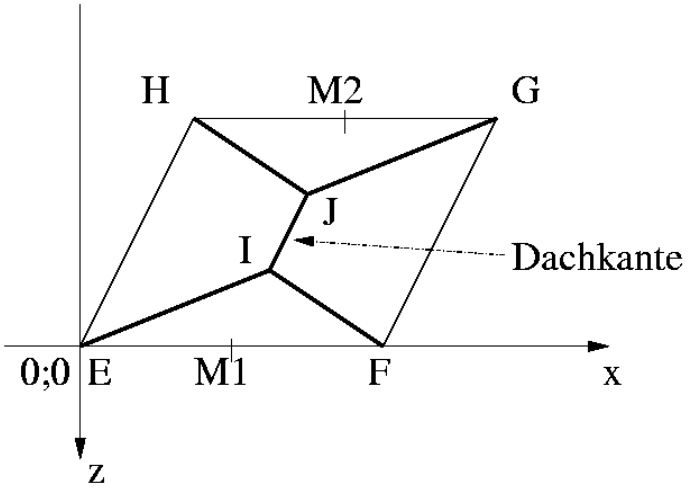



**Abbildung 5.9**: Beschriftung der Dachpunkte in 2D

Der durch die Punkte M1 und M2 definierte Vektor dient als Richtungsvektor mit der Haustiefe als Länge. Dieser Richtungsvektor wird mit dem jeweiligen, errechneten Verhältnis zur Haustiefe multipliziert und mit dem Punkt M1 addiert, sodass zwei neue Punkte I' und J' entstehen. Abschließend wird die Dachhöhe zu I' und J' addiert.

Im Folgenden wird erläutert, weshalb nur Häuser, deren Grundfläche ein Parallelogramm ist, erstellt werden.
Bei Neubauten kann man meist davon ausgehen, dass sie rechwinklige Hausecken besitzen. Die Gebäude der Osnabrücker Altstadt sind allerdings vor einigen Jahrhunderten erbaut worden. Selbst wenn sie damals im rechten Winkel errichtet worden wären, wobei dies aufgrund der Grundstückaufteilung selten der Fall war, befänden sie sich heute nicht mehr in der ursprünglichen Form. So kommt es dazu, dass einige Häuser schief stehen.
Zunächst gab es den Gedanken auch „schiefwinklige" Häuser zu erstellen. Jedoch scheiterte dies aufgrund der Dachverhältnisse. Soll die Dachkante parallel zur Grundebene verlaufen, ist es schwierig oder unmöglich, geeignete Dachseiten zu erstellen, da die Dachseiten nicht immer als Ebene darstellbar sind. Sobald die Grundfläche kein Parallelogramm ist, lässt sich keine zur Grundfläche parallele Dachkante erstellen.
Um ein Dach erstellen zu können ist es nötig, dass der Anwender die Höhe des Daches und die Dachkantenlänge angibt. Um Häuser mit beliebiger Grundfläche zu erstellen, ist es außerdem notwendig vier Grundkoordinaten anzugeben. Somit sind die Dachseiten eines Hauses in Form von `IndexedFaceSets` durch jeweils vier vorgegebene Punkte definiert, diese ergeben sich aus den Haushöhen, der Grundkoordinaten und der Dachkante. Eine Ebene wird allerdings durch drei Punkte definiert. Nun kann es vorkommen, dass nicht alle vier Punkte des `IndexedFaceSets` der Dachseite in einer Ebene liegen. Die folgende Abbildung dient zur Darstellung der möglichen Dachebenen. In der linken Abbildung liegt Punkt G in der Ebene, die durch die Punkte B, C und F verläuft. In der rechten Darstellung liegt Punkt G' jedoch nicht in der von den Punkten B, C und F definierten Ebene.

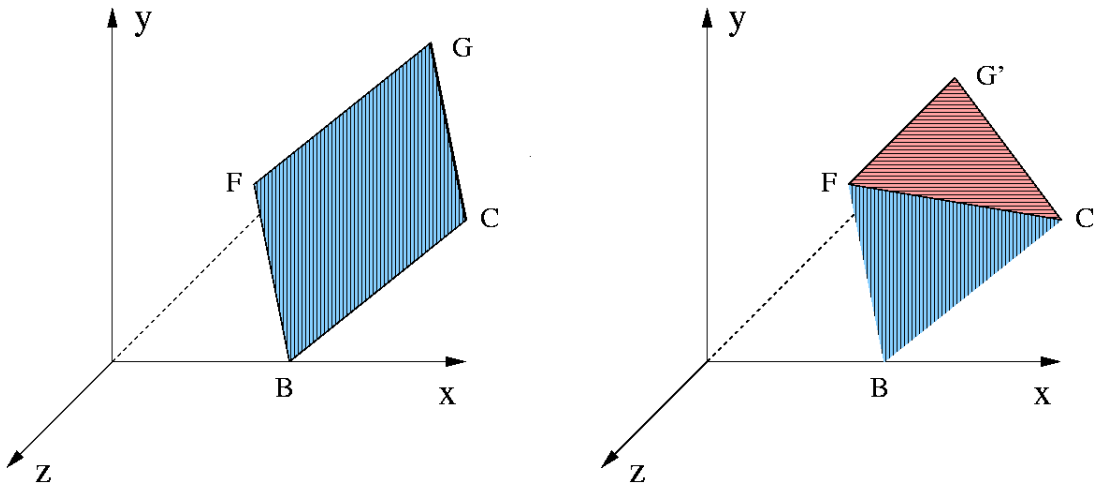



**Abbildung 5.10**: Darstellung eventueller Dachebenen

Um diesem Problem entgehen zu können, wäre bei der Dateneingabe eine Menge an zusätzlichen Einschränkungen erforderlich. Dies würde den Anwender allerdings wieder sehr einschränken und die Eingabe sehr verkomplizieren. Schließlich ist es benutzerfreundlicher und dennoch nicht sehr funktionseinschränkend, die Grundflächenform auf Parallelogramme einzuschränken.

### 5.3.4 Zusammenstellung der Objekte zu einer VRML-Welt

Einfache Objekte wie Häuser oder `Billboards` können im Szenegraphen untereinander geschrieben werden. Hierzu werden die VRML-Dateien aller in der Welt enthaltenen Objekte, die keine weiteren Funktionen wie Animation oder Interaktion enthalten, in die VRML-Datei der Welt kopiert.

*Besonderheiten beim Anlegen von Interaktionen oder Animationen:*

Sollen Interaktionen oder Animationen in eine 3D-Welt eingebaut werden, so treten einige Schwierigkeiten auf.

Bei einer Animation, wie zum Beispiel `Sound`, wird ein `TimeSensor` (siehe Kapitel 2.1.3.2 c) und 2.1.3.3 d)) benötigt, um eine Beziehung zwischen dem Benutzerstandort in der 3D-Welt und der Lautstärke der Musik anzugeben. Sobald

der Benutzer in den Bereich der Musik eintritt, schickt der Sensor über die Route eine Anweisung, die Welt mit Musik zu unterlegen. Um den TimeSensor möglichst optimal zu gestalten, wird ein Prototyp erstellt. Allerdings werden Prototypen zu Anfang, also noch vor dem Szenegraphen eines VRML-Quelltextes angelegt. Erst nach dem Szenegraphen stehen die Anweisungen der Route.

Interaktionen verursachen weitere Problemstellungen. Zusätzlich zu den Sensoren, hier TouchSensoren, werden Trigger, sowohl Integer- als auch BooleanTrigger, benötigt.

Die tatsächliche Definition eines TouchSensors steht allerdings in derselben Gruppe in der das zu klickende Objekt steht. Somit darf das Objekt nicht mehr nur aus einer anderen VRML-Datei kopiert werden, sondern muss vor dem Einfügen in den Quelltext der Welt verändert werden, um den Sensor an der richtigen Stelle einzufügen.

Es wird also für jedes zu klickende Objekt ein TouchSensor in derselben Gruppe definiert. Außerdem wird ein neues Objekt in Form eines HUDs erzeugt, um ein in der Welt feststehendes Textfeld zu erzeugen. Dieses Textfeld besitzt ebenfalls einen TouchSensor, um bei einem Klick auf sich unsichtbar werden zu können. Der VRML-Knoten HUD und Beispiele hierzu werden im Kapitel 2.1.3.3 d) erläutert. Der Quelltext für das interaktive Textfeld kann natürlich im Bereich des Szenegraphens stehen.

Ein größeres Hindernis, als die Objekte und Sensoren für eine interaktive Gruppe zu erstellen, liegt allerdings darin, die notwendigen Trigger und Routes aufzustellen. Denn zu jedem neuen interaktiven Objekt müssen Route-Anweisungen hinzugefügt werden, sodass das neuhinzugefügte Textfeld unsichtbar wird, wenn auf ein anderes interaktives Objekt geklickt wird, um dessen Textfeld hervorzuheben. Außerdem werden im Gegenzug Routes beigefügt, die alle anderen Textfelder verschwinden lassen, falls auf das Objekt des neu hinzugefügten Textfeldes geklickt wird.

Diese Problemstellungen wirken sich auf den Aufbau einer VRML-Datei in folgender Hinsicht aus. Am Anfang des Quelltextes stehen die Prototypen, falls

`TimeSensoren` oder `Trigger` benutzt werden. Daraufhin werden in den Szenegraphen alle Objekte kopiert, die keinerlei Interaktionen oder Animationen besitzen. Anschließend werden notwendige `HUDs` beschrieben und dann die dazugehörigen klickbaren Objekte und deren `Sensoren`. Als nächstes werden die `Sound`-Knoten und deren `Sensoren` in den Quelltext geschrieben. Bevor schließlich die `Route`-Anweisungen ermittelt werden, werden alle notwendigen `Trigger` definiert.

Um die Berechnungen beim Anlegen von Interaktionen und Animationen nicht zu sehr auszuweiten und um einen sauberen Programmfluss zu garantieren, können Interaktionen und Animationen nur in Welten hinzugefügt werden. Zum einen wird der Speicherplatz sonst für die Dateien sehr groß und zum anderen können die Berechnungen, um später mehrere animierte oder interagierende Objekte zu einer Welt zusammenzufügen, sehr viel Zeit in Anspruch nehmen. Eine Ausweichmöglichkeit ist, ein einzelnes Objekt in einer Welt zu definieren, so kann der Anwender auch zu diesem Objekt Animation und Interaktion hinzufügen.

## 5.4 Erstellen einer Welt

Textdateien herkömmlicher Objekte enthalten Daten über den Namen und die Beschaffenheit des Objekts. Wird eine Welt erzeugt, so werden in der Textdatei nicht nur der Name und die Größe der Welt gespeichert, sondern auch die Namen der in ihr enthaltenen Objekte.
Die Größe der Welt ist veränderbar. Dies ist zum Beispiel notwendig, falls eine Welt in eine andere Welt eingefügt werden soll. Es könnte sein, dass in diesem Falle die Größe der beiden Welten nicht übereinstimmt, sodass die Objekte in der einen Welt kleiner sind als in der anderen. Um nicht die Objekte der ganzen Welt ändern zu müssen, ist von großem Vorteil, die Größe der Welt zu ändern.

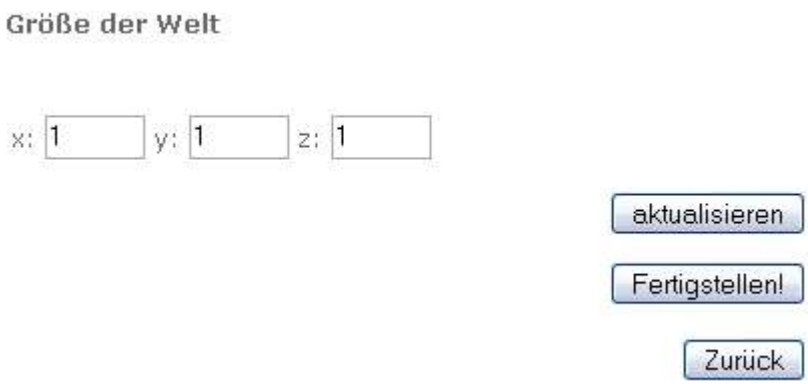



**Abbildung 5.11**: Eingabeformular, um die Weltgröße zu ändern

### 5.4.1 Abspeichern und Verhalten enthaltener Objekte

Die enthaltenen Objekte werden in änderbare und nicht änderbare unterteilt. Die ursprünglichen Text- und VRML-Dateien der änderbaren Objekte werden kopiert, so können nicht nur die in der Welt enthaltenen änderbaren Objekte verändert werden, ohne dass dieses Auswirkungen auf die ursprünglichen Objekte hat, sondern auch die ursprünglichen Objekte ohne weitere Auswirkungen verändert werden. Dies ist von großer Bedeutung, sobald ein Objekt in zwei verschiedenen Welten enthalten ist und nachträglich verändert werden soll.

Außerdem werden in der Textdatei Angaben über die enthaltenen TextSensoren und Sounds gemacht. Hierzu werden lediglich die Namen der Textsensoren oder Sounds gespeichert. In den Textdateien der jeweiligen Sensoren stehen die zusätzlichen veränderbaren Eigenschaften eines jeden Sounds bzw. TextSensors, wie der anzuzeigende Text und seine Eigenschaften oder die Objekte, die anklickbar sind um das Textfeld erscheinen zu lassen. TextSensoren oder Sounds haben demnach keine eigenen VRML-Dateien, da diese Eigenschaften einer 3D-Szene nur in einer Welt vorkommen und nicht alleine im Raum stehen.

### 5.4.2 Löschen eines Objekts aus einer Welt

Ein in einer Welt enthaltenes Objekt kann auf zwei verschiedene Arten aus dieser Welt wieder entfernt werden.

Es ist möglich, das Objekt vollkommen vom Server zu löschen. In diesem Falle werden alle Dateien, die zu diesem Objekt gehören, gelöscht. Hierbei müssen drei Fälle unterschieden werden. Ein Objekt kann unveränderbar sein, also keine Textdatei besitzen. Es kann änderbar oder sogar eine Welt sein, bei der auch alle in ihr enthaltenen Objekte gelöscht werden müssen.

Außerdem kann ein Objekt aus der Welt entfernt werden und trotzdem unter anderem Namen auf dem Server verbleiben. Um dies zu erreichen, werden Dateien, die zum entfernten Objekt gehören, unter einem neuen Namen abgespeichert.

In beiden Arten des Löschens wird das entfernte Objekt aus der Textdatei der Welt gelöscht und die VRML-Datei der Welt neu geschrieben.

### 5.4.3 Abspeichern und Verhalten einer Welt in einer anderen Welt

Eine Welt kann wie schon erwähnt andere Welten enthalten. Hierzu wird in der Textdatei der Oberwelt (Welt A) die einzufügende Welt (Welt B) als änderbares Objekt gespeichert. Außerdem werden, wie bei änderbaren Objekten auch, die Dateien aller Objekte, die in der Welt B enthalten sind, in Dateien der Welt A kopiert.

Eine Besonderheit ist der Umgang mit Textfeldern. Textfelder stehen immer in Bezug zur Oberwelt, denn die anklickbaren Objekte können aus verschiedenen Welten zusammengesetzt werden. Somit werden Textfelder nur in der Oberwelt definiert und aus der eingefügten Welt gelöscht. Das bedeutet, dass, wenn eine eingefügte Welt später aus der Oberwelt entfernt und auf dem Server weiterhin gespeichert wird, die entfernte Welt keine Textfelder mehr besitzt.

## 5.5 Eingabedaten in Form von Textdateien

Wenn dem Anwender die Eingabe der einzelnen Daten für ein Haus zu mühselig ist, so hat er die Möglichkeit, die Textdatei eines bestimmten herkömmlichen Objekts selbst zu erstellen. Diese Textdatei kann der Benutzer hochladen, gegebenenfalls mithilfe der Applikation verändern und eine VRML-Datei dazu erstellen lassen.

Zurzeit ist es nicht möglich, in einer Textdatei Daten mehrerer Häuser abzuspeichern und hochzuladen. Dies würde es zunächst erleichtern, die Häuser zu erbauen. Da jedoch die Häuser in der Osnabrücker Altstadt zu individuell sind, ist kaum zu umgehen, die Texturen oder die Form der Häuser im Nachhinein zu verändern.

Es ist bisher auch noch nicht möglich, eine eigene Welt als Textdatei hochzuladen. Zum einen bringt diese Art, eine Welt zu erstellen, viele Fehlermöglichkeiten mit sich, beispielsweise müssen alle Namen in der richtigen Form geschrieben werden. Zum anderen müssten für größere Welten die Textdateien aller in ihr enthaltenen Objekte und deren Texturen hochgeladen werden. Dies würde einen enormen Aufwand bedeuten. Für den Benutzer soll es außerdem im Verborgenen bleiben wie die Textdatei einer Welt aufgebaut ist.

## 5.6 Hochladen und herunterladen von Dateien

Die Applikation bietet an, Dateien des Anwenders hochzuladen. Nicht nur eigene VRML- oder Textdateien, sondern auch zusätzliche Texturen oder Musikquellen können abgespeichert werden. Diese Dateien müssen unter einem Namen abgespeichert werden, der noch nicht in dem Ordner des Anwenders enthalten ist.
Grafiken werden in den Formaten JPG, PNG oder BMP unterstützt. Musikquellen sollten in Wav oder Midi angegeben werden.
Der Benutzer kann die von ihm angelegten VRML-Dateien als tar-Datei herunterladen. Wenn der Anwender ein Objekt ausgewählt hat, dass er herunterladen

möchte, wird ein tar-Archiv angelegt, in dem die VRML-Datei des Objektes und die dazugehörigen Ordner enthalten sind. Dieses Archiv wird dem Benutzer zum Herunterladen angeboten.

# III. Abschließende Betrachtungen

# 6 Fazit

Die im Laufe dieser Bachelorarbeit entstandene Applikation bietet dem Anwender ein Tool, mit dem er den Grundbaustein für eine interaktive, multimedial animierte, virtuelle 3-dimensionale Stadt erstellen kann. Wie in einer realen Stadt differenzieren sich auch in der Applikation verschiedene Objektarten.

Zunächst ist es möglich, die Umwelt der 3D-Szene zu definieren. Hierbei können Größe und Textur der Bodenfläche, Art und Standort des Lichtes, Texturen der Umgebung, wie zum Beispiel ein Himmel oder auch die Position des ersten Benutzerstandortes vom Anwender variabel festgelegt werden. Außerdem können mithilfe verschiedener Eingabedaten Personen, Bäume, aber auch einzelne Wände, die zum Beispiel zur Begrenzung der 3D-Stadt dienen, erzeugt werden. Besondere Eigenschaften der Applikation sind das Anlegen von interaktiven Textfeldern oder multimedialer Animation. So kann der Gang durch die Osnabrücker Altstadt interessanter gestaltet werden. Möchte der Benutzer nähere Informationen zu einem Objekt erhalten, kann auf dieses geklickt werden, um in der rechten oberen Ecke ein Textfeld erscheinen zu lassen. Während des weiteren Stadtrundgangs bleibt dieses Textfeld in seiner festen Position bestehen, bis der Benutzer durch einen weiteren Klick wieder verschwinden lässt.

Eine für den Benutzer angenehme Eigenschaft ist die Möglichkeit, die 3D-Szene mit verschiedenen Musikstücken zu unterlegen. Der Anwender kann die Anzahl der Musikstücke, deren Lautstärken und Standorte festlegen. Musik ist dann in verschiedenen Regionen der Stadt zu hören, sobald der Benutzer diese betritt.

Schließlich kann der Anwender die von ihm angelegten oder Default-Objekte in einer Welt zusammenfassen.

Sollte der Anwender spezielle Objekte, die nicht in der Applikation verfügbar sind, in die 3D-Welt hinzufügen wollen, kann er deren VRML-Dateien selbst erzeugen, diese hochladen und in eine Welt einbetten.

Alle vom Benutzer in der Applikation definierten Objekte können nachträglich verändert werden, selbst wenn diese in eine Welt eingebunden wurden.
Selbstverständlich kann der Anwender die einzelnen Objekte auch vom Server löschen. Wurde ein Objekt einer Welt zugefügt, kann dieses aus der Welt gelöscht werden, indem das Objekt entweder vollständig vom Server gelöscht, oder nur aus der Welt und nicht vom Speicher des Servers entfernt wird. Weiterhin bietet die Applikation die Möglichkeit die erzeugten VRML-Dateien als tar-Archiv herunter zuladen. So ist es dem Anwender freigestellt, die VRML-Welt auf verschiedene Art weiter zu verwenden.
Um dem Benutzer die Datenangabe zu erleichtern, werden nur einige Eigenschaften der einzelnen Objektarten variabel gehalten. Auf diese Weise werden vom Benutzer nicht zu viele Daten erfordert, und die Rechenzeit zur Erstellung einer VRML-Datei bleibt gering.

Bei der Erzeugung einer VRML-Datei wird besonders darauf geachtet, die VRML-Dateien möglichst effizient und auf den Benutzer optimal zugeschnitten zu erstellen. Dieses ist ein besonderer Vorteil der Applikation, denn viele 3D-Modellierer erstellen VRML-Dateien, die nicht ausschließlich die vom Anwender gewünschten Eigenschaften beschreiben. Zum Beispiel werden in den von Vizx3D erstellten VRML-Dateien standardmäßig einige Prototypen definiert.

Da die VRML-Datei auf den Benutzer vollständig angepasst wird, ist diese später leicht erweiterbar. So wird es dem Benutzer ermöglicht, die verwendeten Interaktionen und Animationen als Beispielimplementation zu nutzen.

„Stein für Stein“ kann der Anwender seine eigene 3-dimensionale Stadt aufbauen, ohne dass seiner Fantasie Grenzen gesetzt werden.

Ein entstandenes Beispiel der Osnabrücker Altstadt als VRML-Welt ist unter
https://snowball.informatik.uni-osnabrueck.de/ktoepler/Welt/
zu erreichen.

# 7 Aussicht

In dieser Bachelorarbeit wurde eine Applikation zur Erstellung einer virtuellen 3-dimensionalen Szene entwickelt. Obwohl die Applikation die Grundlage bietet, eine multimediale und interaktive 3D-Stadt zu erbauen, gibt es sicherlich Möglichkeiten diese Anwendung zu erweitern. Einige dieser Möglichkeiten werden im Folgenden vorgestellt:

- Unebene Flächen:
  Die Osnabrücker Altstadt ist weitestgehend eben. Es gibt kaum Steigungen oder ähnliches. Dennoch ist es möglich, um die Applikation auch für andere Städte nutzen zu können, diese dahingehend zu erweitern, dass Objekte auch auf schiefem Boden und nicht immer auf der x-z-Ebene stehen.

- Straßenlampen, Schilder etc.:
  Die Applikation bietet neben der Möglichkeit Billboards, also sich dem Benutzer zudrehende Objekte, zu erstellen, auch die Möglichkeit einzelne Wände zu erstellen. Dennoch ist es eine sinnvolle Erweiterung Straßenlampen oder auch Straßenschilder in die Welt einbauen zu können. Diese Objekte müssten mithilfe eines feststehenden IndexedFaceSet, dessen Anzahl der Ecken variabel ist, definiert werden.

- Animationen oder Interaktionen:
  Weiterhin wäre es interessant, auch kompliziertere Objekt mithilfe der Applikation erstellen zu können. So könnten weitere Animationen oder Interaktionen hinzugefügt werden. Zum Beispiel könnte der Benutzer die Tür eines Hauses öffnen und eintreten.

- Bearbeitung von Texturen:
  Die Bearbeitung der Texturen direkt in die Applikation mit einzubinden ist sehr sinnvoll, und hebt die Benutzerfreundlichkeit enorm. So muss der Anwender nicht noch ein zweites Tool benutzen, um seine Texturen zu modifizieren, und er könnte sofort den Effekt der Veränderung auf dem `IndexedFaceSet` begutachten.

Weitere Eigenschaften einer Stadt zu berücksichtigen, ist im Rahmen dieser Bachelorarbeit zu speziell. Allerdings können neue Features oder Figuren leicht hinzugefügt werden. Es gibt sicherlich noch einige Erweiterungen, die als neue Features in einer Stadt dienen könnten.

# A.Literaturverzeichnis


[VRML9701] David R. Nadeau,
Introduction to VRML 97

[VRML9702] The Annotated VRML 97 Reference,
http://accad.osu.edu/~pgerstma/class/vnv/resources/info/AnnotatedVrmlRef/ch1.htm

[VRML9703] Handbuch zu VRML von VRMLsuck v0.93
http://www.neeneenee.de/vrml/desktop.html

[VRML9704] Jörg H. Kloss,
VRML 97 : der internationale Standard für interaktive 3D-Welten im World Wide Web

[VorlCG] Oliver Vornberger,
Vorlesung Computergrafik,
http://www-lehre.inf.uos.de/cg

[WebKon] Homepage des Web3D-Konsortiums,
http://www.web3d.org/

[ISO-STD] Homepage der „International Organization for Standardization“,
http://www.iso.org/iso/en/ISOOnline.frontpage

[PHP01] Referenzhandbuch zu PHP,
http://php3.de/manual/de/

[PHP02] Handbuch zu PHP,
http://www.php-homepage.de/manual/

[PHP03] Tutorial zu SelfPHP,
http://www.selfphp.net/selfphp/

[PHP04] Forum zu PHP,
http://www.hohlerzeh.de/

[HTML] Tutorial zu HTML,
http://de.selfhtml.org/

[JScript] Tutorial zu Javascript,

http://de.selfhtml.org/javascript/

Die Literaturquellen beziehen sich auf gedruckte Literatur und auf Websites. Websites bieten in der schnelllebigen Informatik einen aktuelleren Zugang zu Informationen. Allerdings sind sie schnell veraltet oder können unter Umständen später nicht mehr erreichbar sein.
Bei Abgabe dieser Bachelorarbeit waren alle Adressen aktuell und erreichbar.

# B.Inhalt der CD-Rom

- arbeit — schriftliche Ausarbeitung der Bachelorarbeit
- arbeit/pdf — PDF-Version
- arbeit/doc — WORD-Version

- applikation — PHP-Dateien der zur Realisierung der Applikation

- beispiele — Beispiele
- beispiele/quelltext — VRML- und PHP-Beispiele der Ausarbeitung

- programm — Vizx3D, Version 1.2

# C.Erklärung

Hiermit erkläre ich, die vorliegende Arbeit selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Osnabrück, den 21.10.2005

.....

Katrin Töpler