Begleitmaterial für die Übungen „Einführung in die Shark™-Entwicklungs-Umgebung“

Interaktive virtuelle 3D-Welten: Richtlinien

Axel Hoppe

Style Guide und Tipps für den Designer,
Style Guide für den Programmierer.

Letzte Änderung von 3. April 2005

Inhaltsverzeichnis

1 Motivation

Im Sinne der Erarbeitung eines sinnvollen *Organisatorischen Konzepts* mit Hinblick auf ein gemeinsames Arbeiten mehrerer Leute an einem größeren VR- oder Visualisierungsprojekt gelten die folgenden Richtlinien und Festlegungen für die Vorgehensweisen bei der Modellierung und Texturierung.

Auch wenn die augenblicklich vorliegenden Aufgaben zur Erfüllung eines erfolgreichen Lehrziels für die Veranstaltung als „Einzelkämpfer“ bewältigt werden müssen, bereitet ein konsequentes Einhalten der folgenden Tugenden den Weg für spätere Gemeinschaftsprojekte.

Ähnliches gilt für die Programmierung: Jeder Entwickler hat sich vermutlich im Laufe der Zeit einen persönlichen Stil beim Erstellen von Quelltexten zugelegt, der sich auf persönliche Erfahrungen und Vorlieben gründet. Deshalb ist der Versuch, einen solchen Stil zu vereinheitlichen, etwas „heikel“.

Auch wenn derlei Konventionen mit dem unbequemen Umgewöhnen des persönlichen Stils verbunden sind, fördern sie die folgenden sehr wichtigen Sachverhalte, die gerade für die Teamarbeit unerlässlich sind:

- Einarbeitungsaufwand in die Arbeit anderer,
- Austauschbarkeit zwischen verschiedenen Entwicklern,
- Kompatibilität zu Einstellungen der Editoren und vor allem
- Nachnutzbarkeit.

2 Richtlinien für den Designer

2.1 Tugenden

Die nachfolgenden Richtlinien dienen der Einhaltung der folgenden „Tugenden“ beim Erzeugen der 3D-Szenen:

Übersichtlichkeit. Alle Geometrien, Formen und Materialien werden so erzeugt, dass jeder Teilnehmer sich nach Möglichkeit mühelos in der Szene eines anderen zurechtfindet.

Nachnutzbarkeit. Geometrien werden so erstellt, dass sie später für einen anderen Anwendungsfall eingesetzt werden können.

Nachvollziehbarkeit. Eine erhöhte Nachnutzungsmöglichkeit ergibt sich für Objekte, die sich sehr leicht an eine neue Szene anpassen lassen. Dazu ist es notwendig, dass der Nachnutzende nach Möglichkeit in Teilstufen des erfolgten Konstruktionsprozesses korrigierend eingreifen kann. Anpassungen an neue Gegebenheiten sind dann sehr einfach durch die Modifikation von Parametern möglich.

Kompatibilität. Die Szenen aller Teilnehmer sind beliebig untereinander austauschbar bzw. Teilszenen können ohne Probleme in einer Gesamtszene integriert werden.

Sparsamkeit. Die Erzeugung der Szene erfolgt stets derart, dass nur die für korrekte Umsetzung der Idee notwendigen Ressourcen (z. B. an Polygonen oder Texturgröße) verbraucht.

2.2 Benennungen

Grundsätzlich wird für alles, was in der Szene in einem der verschiedenen Editoren erzeugt wird, eine *Benennung* vergeben. Das dient der Nachnutzbarkeit und vor allem Übersichtlichkeit für eine spätere Exploration der Szene.

2.2.1 Aufbau der Benennung

Die Bennennung für Objekte, Formen, Gruppen und Materialien setzt sich wie folgt zusammen:

$$[Kennung]\_[Name]\langle\_[Funktion]\rangle\langle\_[Nummer]\rangle$$

Zwingend müssen die *Kennung* und der *Name* angegeben werden. *Funktion* und *Nummer* sind optional. Ferner gilt für die Bildung folgendes:

- keine Umlaute oder Sonderzeichen verwenden,
- grundsätzlich alles klein schreiben,
- keine Leerzeichen – Worttrennungen werden durch den Unterstrich „_“ vorgenommen.

Grundsätzlich gilt für die Erzeugung der Benennung: So kurz als möglich, aber so lang als nötig – letztlich sind absolut treffende Benennungen zwar gut für die Nachvollziehbarkeit der Modellierung. Sie mindern aber die Übersichtlichkeit, wenn in der Objektliste mit der beschränkten horizontalen Darstellung nur noch ein Bruchteil der Benennung zu erkennen ist.

2.2.2 Kennung

Die *Kennung* dient dazu nachzuvollziehen, welcher Teilnehmer für die Erstellung eines Szenenteils verantwortlich. Grundlage für die Kennung sind die Initialen des Namens. Bei der Teilnahme an den Übungen wird die Kennung zu Beginn festgelegt.

2.2.3 Name

Der *Name* kennzeichnet den Wesenszug eines Objekts und erklärt natürlichsprachlich, um was es sich hierbei handelt. Der Name kann aus verschiedenen Bestandteilen, durch „_“ getrennt, zusammen gesetzt werden. Für die Bildung eines zusammen gesetzten Namens gilt: Links mit allgemeiner Bezeichnung beginnen, nach rechts spezieller werden.

2.2.4 Funktion

Die Namen von Hilfsobjekten werden mit einem Postfix versehen, der die *Funktion* des Hilfsobjekts kennzeichnet. So wird z. B. bei zusammen gesetzten Objekten bereits aus der Objekt-Namens-Liste ersichtlich, welches Objekt die Form und welches Objekt der Erzeugungspfad ist.

2.2.5 Nummerierung

Objekte mit gleichem Namen werden durchnummeriert. Die *Nummerierung* wird vor allem verwendet, wenn Objekte geklont werden sollen und ermöglicht so die Nutzung des Inkrementierungsmechanismus von MAX. Hierbei gilt:

- Immer mit „`00`“ beginnen,
- immer (mindestens) zweistellig, mit auffüllender „`0`“, also „`00`“, „`01`“, ... „`09`“, „`10`“ etc.
- vom Wortstamm immer mit dem Trennzeichen „`_`“ separieren.

2.2.6 Beispiele

Beispielhaft wird angenommen, der Teilnehmer Thomas Georg Müller erstellt eine Szene. Zu Beginn der Übungen erhält er die Kennung `tgm` zugewiesen. Dann sind folgende Benennungen von Objekten gültige Beispiele:

```
tgm_tisch_bein_00
tgm_laterne_mast
tgm_laterne_mast_form
tgm_laterne_mast_pfad
```

Beispiele für korrekt benannte Materialien:

```
tgm_metall_gusseisen
tgm_glas_laterne
tgm_glas_schaufenster
```

2.2.7 Hilfreiches Werkzeug: „Rename Objects“

In jedem 3D-Editor gibt es Werkzeuge, die eine spätere Korrektur von Benennungen ermöglichen und dabei auch auf mehreren Objekten operieren.

Im MAX wird dies über das Zusatzprogramm „Rename Objects“ vorgenommen. Der Dialog zum Umbenennen von selektierten Objekten wird über

- Menü `Tools`,
- Eintrag `Rename Objects...`

aufgerufen.

2.3 Strukturieren

Die korrekt benannten Objekte werden nach Zusammengehörigkeit gruppiert. Die damit erzeugten Hierarchien sollten aber weitestgehend flach sein; zu viele Gruppierungen erschweren die Bearbeitung, da sie erst geöffnet werden müssen.

2.4 Grundeinstellungen

Für die Kompatibilität der Szenen untereinander werden die folgenden Grundeinstellungen für

- Einheiten,
- Gittersprungfunktion und
- Hilfsmittel für das Verschieben

festgelegt. Die Einstellungen werden in der Szene abgespeichert. Diese Einstellungen sollten aber vor Beginn einer neuen Arbeit stets überprüft werden.

Die Formateinstellungen für die Animation sind optional.

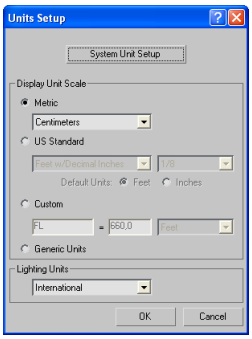


Abbildung 2.1: über `Customize` wird das `Units Setup` aufgerufen.

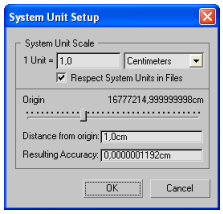


Abbildung 2.2: Im `System Units Setup` einstellen, dass 1 MAX-Einheit `1,0 Centimeter` entspricht.

2.4.1 Einheiten

Das verwendete Einheitensystem wird auf Zentimeter eingestellt – eine Längeneinheit im MAX-Koordinatensystem entspricht 1 Zentimeter. Hierbei müssen zwei Anpassungen gemacht werden:

1. über `Customize` wird das `Units Setup` aufgerufen. Dort für `Display Unit Scale` die Einstellung `Metric` mit der Option `Centimeter` wählen (siehe Abbildung 2.1).

2. In diesem `Units Setup` Dialog wird ferner im Unter-Dialog `System Unit Setup` für `System Unit Scale` eingestellt, dass 1 MAX-Einheit `1,0 Centimeter` entspricht (siehe Abbildung 2.2)

Abbildung 2.3: Aktivierungsflächen für die Gittersprungfunktionen.

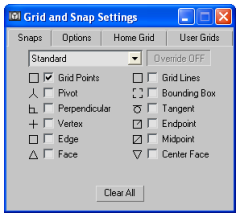


Abbildung 2.4: Aktivieren des Gittersprungs.

2.4.2 Gittersprungfunktionen

Zur Sicherstellung genauen, nachvollziehbaren Modellierens wird u. a. die Gittersprungfunktionalität von MAX benutzen. Vor Beginn der Arbeit an einer 3D-Welt folgendes einstellen:

1. Aktivieren von `Snap`, `Angle Snap` und `Percent Snap` (siehe Abbildung 2.3).

2. Aktivieren der Ausrichtung am Gitter über `Customize` in den `Grid and Snap Settings` die Option `Grid Points` (siehe Abbildung 2.4).

2.4.3 Hilfsmittel für das Verschieben

Das Verschieben kann wahlweise über die `Axis Restrictions` auf bestimmte Achsen beschränkt werden, um den Konstruktionsvorgang zu erleichtern (siehe Abbildung 2.5).

Abbildung 2.5: Die Translation kann auf bestimmte Achsen beschränkt werden.

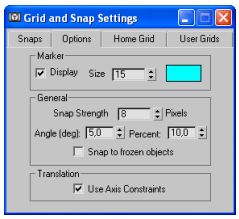


Abbildung 2.6: Aktivieren der `Axis Constraints`.

Diese Einstellung wird über `Customize` in den `Grid and Snap Settings` unter `Options` in der Rubrik `Transaltion` aktiviert (siehe Abbildung 2.6).

2.4.4 Einblenden der relevanten Toolbars

über das Menü `Customize` und das Kommando `Show UI` wird sichergestellt, dass die folgenden Toolbars zum Arbeiten verfügbar sind – dies wird über folgende Einstellungen erreicht:

- `Show Command Panel`,
- `Show Main Toolbar`,
- `Show Track Bar` und
- `Show Tab Panel` – hier wird die Abteilung mit den Achsenbeschränkungen unter die Main Toolbar angedockt – die andere Toolbar kann wieder geschlossen werden.

Um zu vermeiden, dass diese „Floating Toolbars“ versehentlich umkonfiguriert werden können, kann diese Einstellung über das `Customize`-Menü und `Lock UI Layout` fest zementiert werden.

2.4.5 Optional: Formateinstellungen für die Animation

Für das Rendering von Animationen kann es notwendig sein, die Standard-Einstellungen auf das eropäische Format einzustellen, was aber erst beim Videoschnitt interessant ist:

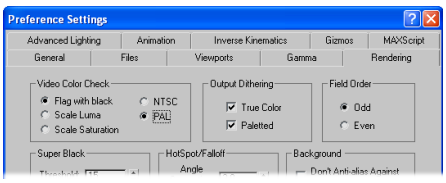


Abbildung 2.7: In den `Preference Settings` unter `Video Color Check` die Einstellungen auf PAL setzen.

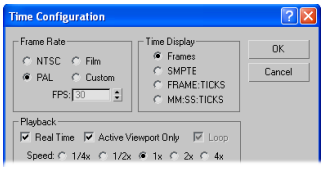


Abbildung 2.8: In der `Time Configuration` für die `Frame Rate` PAL einstellen.

1. In den `Preference Settings` unter `Video Color Check` die Einstellungen auf PAL setzen (siehe Abbildung 2.7).

2. In der `Time Configuration` für die `Frame Rate` PAL einstellen (siehe dazu Abbildung 2.8). Dahin gelangt man schnell, indem mit der rechten Maustaste auf ein Element der Animationssteuer-Element ein Kontextmenü ausgeklappt wird.

2.5 Modellierung

2.5.1 Allgemein

An den Modellen werden nach Abschluss der Arbeiten oftmals noch im Zuge von Optimierungen oder der Nachnutzung in anderen Projekten Veränderungen vorgenommen. Hierbei ist eines der wichtigsten Werkzeuge der Modifyer-Stack der Geometrie. Dieser muss übersichtlich und nachvollziehbar gehalten werden.

Durch wenig diszipliniertes Modellieren kann es z. B. passieren, dass hier Edit-Mesh- oder UVW-Map-Modifyer nacheinander angewendet wurden, wobei der Stack unnötig anwächst und das Editieren verkompliziert wird.

Vor allem prozedurale Modelliertechniken sind zu bevorzugen, da im Zuge eines Op-

timierungsarbeitsgangs Polygone eingefügt oder reduziert werden müssen. Prozedurale Modelliertechniken sind etwa

Polygonerzeugung auf Basis von 2D-Shapes, Modellieren mit 3D-Standardprimitiven, wie Kugel, Zylinder, Plane etc., Modellieren mit zusammengesetzten Objekten, wie z. B. Lofting.

Nicht nachvollziehbar sind z. B. Vorgehensweisen, wo die Ursprungsobjekte und deren Parameter verloren gehen, etwa bei der Funktion `Detach` für ausgewählte Polygone im Modifyer `Edit Mesh`.

2.5.2 Polygonreduziert Modellieren

Nur so viele Polygone verwenden, wie wirklich benötigt werden. Aber: Polygone für Effekte ergänzen:

Für die Ausleuchtung der Szene ist es durchaus notwendig, explizit Polygone zu ergänzen. Bei so genannten Vertex Shadern wird bei der Beleuchtung die Lichtverteilung pro Polygon errechnet. Je feinmaschiger und gleichmäßiger ein Polygongitter ist, um so präziser wird eine Beleuchtungsverteilung dargestellt (z. B. hell in der Mitte unter einer Lampe, dunkel am Rand).

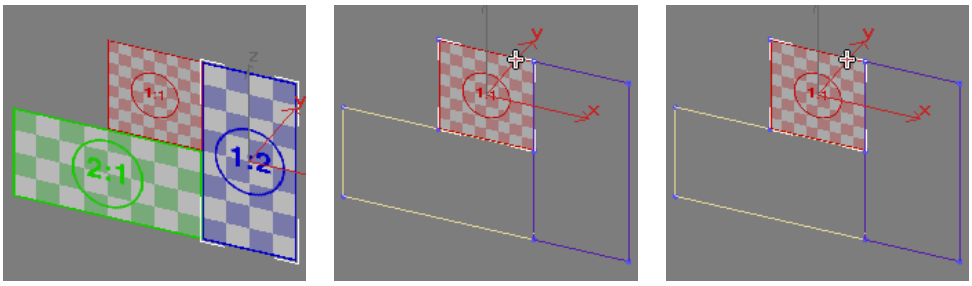
Beispiel: Eine Wartehalle in einem Bahnhof wird modelliert. Die Grundfläche ist rechteckig. Der Fußboden wird explizit mit dem Standard-Primitiv Plane erzeugt. Über die Parameter der `Plane Length Segs` und `Width Segs` kann die Anzahl der Polygone gleichmäßig über die ganz Plane eingestellt werden.

2.5.3 T-Junctions vermeiden

Als *T-Junction* wird eine spezielle Konstellation bei Geometrien bezeichnet, wo ein Eckpunkt eines Polygons auf die Kante eines anderen Polygons trifft – bezogen auf die Kanten ensteht so die Form des Buchstaben „T“,

Hierbei entstehen bei wenig fehlerverzeihenden Bildberechnungsverfahren (vor allem im Echtzeitbereich) Flimmereffekte durch Rundungsfehler auf Subpixel-Ebene.

Abbildung 3a zeigt eine Konstellation von Rechtecken mit verschiedenen Materialien.



(a) Geshadet (b) Einfache Planes (c) Korrigierte Planes

Abbildung 2.9: Beispiel für T-Junctions.

Diese Konstellation muss also auch aus verschiedenen Objekten zusammen gesetzt werden.

In Abbildung 2.9(b) wurde diese Konstellation zunächst aus einfachen, nicht unterteilten Planes zusammen gestellt. Die Eckpunkte des roten Quadrats stoßen hier auf die Kanten des grünen und des blauen Rechtecks – ein klassisches Beispiel für T-Junctions.

Abbildung 2.9(c) zeigt eine Lösungsmöglichkeit: Hier wurde die Unterteilung der blauen und grünen Plane so korrigiert, dass die neu entstandenen Eckpunkte genau auf die Eckpunkte des roten Quadrate fallen.

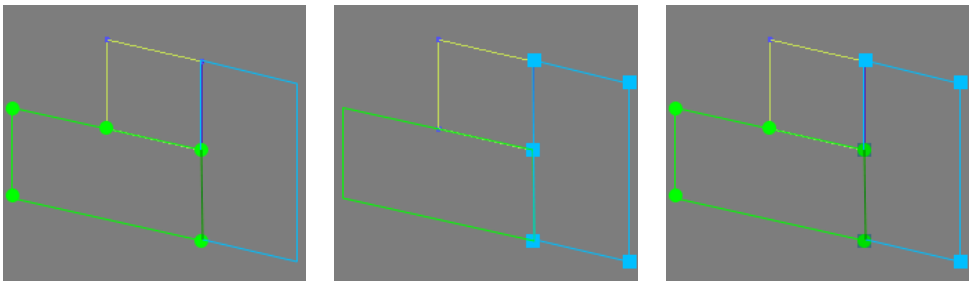
Eine weitere Lösungsmöglichkeit für den vorliegenden Fall besteht darin, das blaue und das grüne Rechteck mit dem Shape Line derart zu erstellen, dass neben den Vertices für die eigentliche Umrandung des Rechtecks auch Vertices an den Stellen gesetzt werden, wo sich Vertices des roten Rechtecks befinden (zur Illustration siehe Abbildung 2.10).

2.6 Vorschau-Texturierung

2.6.1 Motivation

Bereits während der Modellierung werden zwei ansonsten sehr zeitraubende Aufgaben des Texturierers erledigt:

- die Vergabe der Texturkoordinaten und



(a) Line für grünes Rechteck (b) Line für blaues Rechteck (c) zur Vermeidung von T-Junctions

Abbildung 2.10: Vermeidung der T-Junctions mittels Modellierung mit Shapes.

- das Anlegen des Materials.

Damit muss sich der Texturierer nicht mühsam durch die Bestandteile der Szene klicken und sich dabei erst in die vorliegenden Geometrien „hineindenken“.

Die verwendeten Test-Texturen zeigen dem Modellierer an, wie gut und sinnvoll die Texturkoordinaten vergeben wurden und erleichtern so Korrekturen.

Die Test-Texturen sind so vorbereitet, dass ihre Pixel-Auflösungen in Zweierpotenzen vorliegen – dies stellt die Kompatibilität für eine Nutzung der Szene innerhalb eines Echtzeit-3D-Projekts sicher.

Die verwendete Test-Textur zeigt dem Texturierer an, in welchen Dimensionen er die Zieltextur letztlich anlegen muss.

Später muss der Texturierer nur noch die bereits vom Modellierer korrekt benannten Kopien der Test-Texturen im Textur-Verzeichnis durch die Zieltexturen ersetzen.

2.6.2 Vorgehensweise

Die Test-Texturen können unter `http://www.shark3d-user.de/vl/texture.zip` bezogen werden. Die Texturdateien sind gemäß ihres Seitenverhältnisses benannt.

Jedes Objekt wird bereits bei der Modellierung mit Texturkoordinaten versehen – falls die Optionen für die Erzeugung der Texturkoordinaten beim Generieren des Objekts

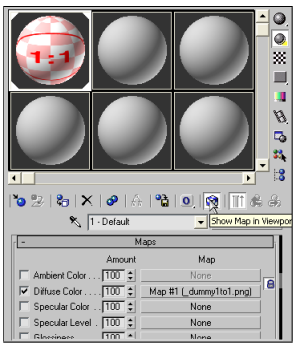


Abbildung 2.11: Option „Show Map in Viewport“.

nicht befriedigend sind, erledigt das der Modifyer `UVW Map`.

Anschließend wird im Material-Editor ein Material der Kategorie „Standard“ für das Objekt erzeugt und mit einem sinnvollen Namen versehen.

Für das Material wird gemäß der gewählten Texturkoordinaten eine passende Test-Textur ausgewählt.

Von dieser Test-Textur wird im Textur-Verzeichnis eine Kopie erstellt und gemäß des Materialnamens benannt.

Die Textur wird als `Diffuse Color` Map zugewiesen.

Zur überprüfung der Texturkoordinaten auf dem Modell kann im Material-Editor die Option „Show Map in Viewport“ aktiviert werden (siehe Abbildung 2.11).

2.7 Tipps für den Umgang mit MAX

2.7.1 Allgemeine Tipps

- falls MAX nicht so zu reagieren scheint, wie es soll: `Esc` hilft,
- Ansonsten sicherstellen, dass keine Unterauswahl in einem Modifyer aktiv ist (Unterauswahl ist dann gelb unterlegt),

Abbildung 2.12: Einstellen des Weltkoordinatensystems.

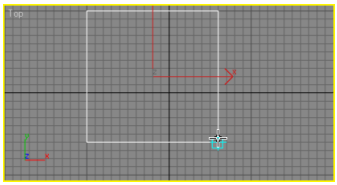


Abbildung 2.13: Vorgehensweise beim Erzeugen eines 3D-Primitivs.

- Tastatur-Eingaben immer mit `Return` abschließen.
- Für alle Transformationen und für das Alignment-Werkzeug gelten: Am besten am Anfang immer mit Weltkoordinaten arbeiten – die Koordinatensysteme können über die Auswahlliste „Reference Coordinate System“ eingestellt werden (siehe Abbildung 2.12).
- Objekte am besten immer mit dem Alignment-Werkzeug ausrichten.

2.7.2 Modellieren mit 3D-Primitiven

Beim Modellieren mit 3D-Primitiven ist ausschlaggebend, auf welche Art und Weise das Objekt erzeugt wurde und wie das lokale Koordinatensystem (repräsentiert durch den Object Pivot) angeordnet ist. Es empfiehlt sich, zu Anfang alle Objekte immer in der „Top View“ zu erzeugen (siehe Abbildung 2.13). Hierbei immer nach dem folgenden Schema vorgehen:

- Nach dem ersten Klick für den Querschnitt des neuen Objekts Länge und Breite festlegen.
- Mit dem zweiten Klick die Höhe des Objekts.

Abbildung 2.14: Werkzeug „Restrict to X“.

2.7.3 Achsen-Restriktionen

Es folgt ein kurzes Beispiel zur Arbeit mit den Achsen-Constraints:

- 3D-Primitiv „Box“ erstellen.
- Nach Vorschrift benennen.
- Werkzeug „Restrict to X“ auswählen (siehe Abbildung 2.14).
- Werkzeug „Select and Move“ auswählen.
- Beispielhaft in Ansicht „Top“ bewegen.

2.7.4 Gruppierungen

Selektierte Objekte werden über das Menü mit

`Group`, `Group`

zu einer Gruppe zusammengefasst. Die Unterobjekte in dieser Gruppe können jetzt nicht mehr separat selektiert werden. Um diese weiterhin bearbeiten zu können und ohne die Gruppe dabei wieder aufzulösen, muss die Gruppe über

`Group`, `Open`

für das Editieren der Unterobjekte geöffnet und über

`Group`, `Close`

wieder geschlossen werden.

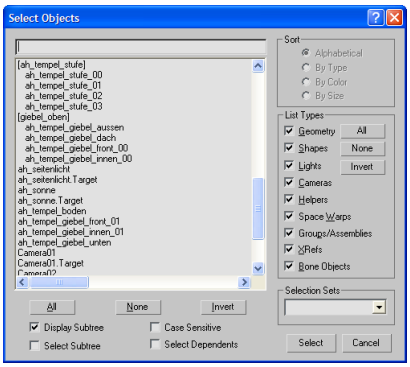


Abbildung 2.15: Gruppierungen in der `Object List`.

Zur besseren Übersicht über die Gruppenstruktur kann die `Object List` benutzt werden (siehe Abbildung 2.15).

2.7.5 Ausgewählte Tastatur-Kürzel

Es folgt eine Auswahl an standardmäßig verfügbaren Tastatur-Kürzeln im 3D Studio MAX zur Beschleunigung der Arbeit:

- `Alt` + `a` „Align“-Werkzeug
- `Alt` + `w` Umschalten zwischen Ein- und Mehr-Ansichten-Modus
- `a` Winkelsprung
- `b` Ansicht „Bottom“
- `f` Ansicht „Front“
- `g` Gitteranzeige
- `h` Dialog „Hide by Name“
- `h` Objekt-Liste
- `l` Ansicht „Left“

- `m` Material-Editor
- `n` Modus „Auto Key“ zur Animations-Aufzeichnung
- `r` Ansicht „Right“
- `s` Gittersprung
- `t` Ansicht „Top“
- `u` Ansicht „Isometric View“

3 Richtlinien für den Programmierer

3.1 Allgemeine Richtlinien

Für das Erstellen von Quelltexten gelten die folgenden allgemeinen Richtlinien:

- Niemals Tabulatoren verwenden (den verwendeten Editor entsprechend konfigurieren).
- Einrückungen werden durch 4 Leerschritte vorgenommen.
- Eine Zeile hat maximal 80 Zeichen. Wird diese Länge überschritten, wird sinnvoll und ästhetisch umgebrochen. Siehe hierzu Vorschläge der enstsprechenden Empfehlungen für die jeweiligen Programmiersprachen (für z. B. Java aus [Mic04, Abschnitt 4.2, „Wrapping Lines“]).

3.2 Richtlinien für Java-Klassen

3.2.1 Grundlegende Philosophie

Die Java-Klassen genügen den vor allem dem Style Guides für Java-Quellcode [Mic04]. An dieser Stelle soll vor allem auf die folgenden „literarischen“ Komponenten dieses Style Guides hingewiesen werden:

- Klammersetzung:
  - Zwischen öffnender und schließender Klammer und dem Parameter steht ein Leerschritt.
  - Folgen mehrere öffnende oder schließende Klammern aufeinander, entfällt der Leerschritt, um Platz zu sparen.

Das Listing 3.1 zeigt mehrere Beipsiele.

```
public int setIndex( int index ) {
    ...
}
if (( i < numElements ) && ( i > minValue )) {
    ...
}
while ( i < maxLetters[ index ]) {
    ...
}
```

Listing 3.1: Beispiele für die Setzung der Klammern.

- Operatoren sind zur besseren Übersicht von Leerschritten eingefasst. Korrekte Beispiele sind im Listing 3.2 aufgezeigt.

```
value = x * 3.0f / y - 7.0f;
if (( i == 3 ) || ( z < maxValue )) ...
```

Listing 3.2: Beispiele für die Platzgestaltung bei Operatoren.

- Nach Kommata oder Semikolons steht ein Leerschritt. Beispiele hierfür werden im Listing 3.3 gegeben.

```
addValues( 2, 7, 15 );
for ( int i = 0; i < numEntries; i++ ) {
    ...
}
```

Listing 3.3: Beispiele für die Platzgestaltung bei Kommata und Semikolons.

Inhaltlich gilt folgendes für den Aufbau von Java-Klassen:

- Die Java-Klassen sollten kurz und übersichtlich bleiben – und nur die eng abgegrenzte Aufgabe abdecken, für die sie gedacht sind. Faustregel: Eine Skript sollte nicht länger 200–300 Zeilen sein.

- Desgleichen sollten die Member-Funktionen kurz sein und mit aussagekräftigen Namen versehen werden: Der Name der Funktion sollte einen so aussagekräftige Namen haben, dass kein zusätzlicher Kommentar notwendig ist.

- Die Anzahl der Member-Variablen sollte möglichst klein sein.

Die weiteren Ausführungen beziehen sich vor allem auf Empfehlungen für ein einheitliches Aussehen und eine einheitliche Vorgehensweise bei der Fehlersuche und Arbeit mit dem Java-Logging der Shark™-Entwicklungs-Umgebung.

3.2.2 Aufbau einer Klasse

Folgende Reihenfolge beim Aufbau der Java-Klasse ist zu berücksichtigen:

1. Header.
2. Alle überladenen „call()"-Funktionen der Superklasse „`Script`", da sie das Interface bilden.
3. Alle **public** deklarierten Member-Variablen (insofern notwendig).
4. Alle **public** deklarierten Methoden.
5. Alle **private** deklarierten Member-Variablen.
6. Der Konstruktor „initActor()".
7. Alle **private** deklarierten Member-Funktionen.

3.2.3 Header

Im Header werden Copyright- und Versionskontroll-Informationen untergebracht. Hierbei z. B. sind der Name des aktuellen Projekts und ein optionaler Kommentar, der die Klasse näher erklärt unterzubringen. Im Normalfall ist das nicht notwendig, der klug gewählte Name der Klasse deren Zweck bereits ausreichend erklärt.

Ein Beispiel für den Header zeigt Listing 3.4.

3.2.4 Variablennamen

Variablen haben aussagekräftige Namen, die sofort den Zweck der Variablen erschließen. Referenzen auf Actoren sollten nicht einfach „`actor`" benannt werden, sondern den Actor-

```
1 ////////////////////////////////////////////////////////////
2 // (c) 2002--2005 Team Uebungsleiter
3 // Projekt Virtuelles Haus, Setzen des Namensschildes
4 // ------------------------------------------------------------
5 // $Id: $
6 // $Source: $
7 ////////////////////////////////////////////////////////////
```

Listing 3.4: Beispiel für einen Header.

```
1 private final int INPUT_NORMAL   = 1;
2 private final int INPUT_MENU     = 2;
3 private final int INPUT_DISABLED = 3;
4
5 private int m_stateInput = INPUT_NOMAL;
```

Listing 3.6: Beispiel für die Verwendung von Zustandsvariablen.

Typ bezeichnen (z. B. „`link`“) oder damit beginnen. Wahlweise kommt auch der Name des Actors in Frage, falls er einen besitzt. Dieser sollte aber ebenfalls mit dem Actor-Typ beginnen.

Member-Variablen beginnen mit einem „`m_`“. Beispiele finden sich im Listing 3.5:

```
1 private Actor m_extconfig;
2 private String m_logId;
```

Listing 3.5: Beispiel für die Verwendung von Member-Variablen.

3.2.5 Benutzung von Zustands-Variablen

Wechselt ein Skript zwischen Zuständen, sollten nicht mehrere Variablen vom Typ **boolean** verwendet werden, um den Zustand zu definieren, sondern eine Zustandsvariable. Ein Beispiel hierfür zeigt Listing 3.6.

3.2.6 Benutzung der Logging-Funktionalität

Beim Logging über die Member-Funktion „reportInfoText()“ wird mit jeder Meldung die *ID* der Klasse mit ausgegeben.

Empfohlene Vorgehensweise: Die im Listing 3.7 dargestellte Methode „myLog()“ implementieren und anstatt der Methode „reportInfoText()“ für Log-Datei-Ausgaben benutzen; ferner die Member-Variable „`m_logId`“ wie im folgenden Abschnitt 3.2.7 beschrieben setzen.

```
1 private void myLog( String msg ) throws Msg
2 {
3     reportInfoText( null, m_logId + ":␣" + msg );
4 }
```

Listing 3.7: Beispiel für die Verwendung der Logging-Funktionalität.

3.2.7 Der Konstruktor

Im Konstruktor der Klasse wird zur Erleichterung der Fehlersuche der Aufruf des Konstruktors in der Log-Datei eigetragen. Das im Listing 3.8 dargestellte Template kann für alle Java-Klassen benutzt werden.

```
1 protected void initActor() throws Msg
2 {
3     m_logId = getClass().getName();
4     myLog( "init" );
5 }
```

Listing 3.8: Beispiel für den Konstruktor.

3.2.8 Logging beim Zugriff auf Shark™-Actors

Wird in der Java-Klasse auf Shark™-Actors zugegriffen, wird für eine bessere Fehlersuche beim Aufruf der Methode „findActor()“ ebenfalls die *ID* der Klasse übergeben. Es empfiehlt sich, standardmäßig die im Listing 3.9 dargestellte Member-Funktion zu implementieren und zu benutzen.

```
1 private Actor myFindActor( String name ) throws Msg
2 {
3     return findActor( m_logId, name );
4 }
```

Listing 3.9: Beispiel für den Konstruktor.

```
1  public void call( String info, Actor sender, String cmd ) throws Msg
2  {
3      myLog( "cmd␣=␣" + cmd + "␣sender␣=␣" + sender );
4      if ( cmd.equals( "play_next_phase" ))
5          playNextPhase();
6      else
7          super.call( null, sender, cmd );
8  }
9  ...
10 private void playNextPhase() throws Msg
11 {
12     ...
13 }
```

Listing 3.11: Beispiel für den Konstruktor.

3.2.9 Reagieren auf Shark™-Events („call()“-Methoden)

Beim Überladen der „call()“-Methoden werden zur besseren Fehlersuche auch Kommandos und Parameter in die Log-Datei geschrieben (siehe Listing 3.10).

```
1  public void call( String info, Actor sender, String cmd ) throws Msg
2  {
3      myLog( "cmd␣=␣" + cmd + "␣sender␣=␣" + sender );
4      super.call( null, sender, cmd );
5  }
```

Listing 3.10: Beispiel für den Konstruktor.

Die „call()“-Methoden sollten keine Programmlogik enthalten, sondern das Kommando in eine Methode mappen, deren Name und Parameter sich aus dem Kommando ergeben.

Also wird für jedes Kommando eine Methode mit gleichem Namen implementiert, das auf das Kommando regiert und den ggf. mitgeschickten Parameter verarbeitet.

Diese Vorgehensweise wird konsequent verfolgt, auch wenn die Methode nur wenige Zeilen enthalten wird. Die „call()“-Methode bleibt dann eine übersichtliche Liste, in der schnell geblättert werden kann.

Ein entsprechendes Beispiel zeigt Listing 3.11.

Kommandos, die von außen an die Klasse geschickt werden werden von innerhalb der Klasse mit „evtInsert()“ geschickten Kommandos unterschieden, indem interne Kom-

mandos immer mit dem Unterstrich „_“ beginnen. Ein Beispiel für die Implementierung eines internen Kommandos zeigt Listing 3.12

```
1 public void call( String info, Actor sender, String cmd ) throws Msg
2 {
3     myLog( "cmd␣=␣" + cmd + "␣sender␣=␣" + sender );
4
5     if ( cmd.equals( "_set_texture" ))
6         setTexture();
7     else
8         super.call( null, sender, cmd );
9 }
```

Listing 3.12: Beispiel für den Konstruktor.

3.3 CD-Dateien

Die CD-Dateien sollten grundsätzlich mit dem Komponenten-Editor editiert werden – das verhindert Syntax-Probleme. Für die Notation gilt folgendes:

- Einzige zulässige Sprache für Namen und Kommentare ist Amerikanisches Englisch.
- Es werden nur Kleinbuchstaben verwendet.
- Actors haben aussagekräftige Namen, die sofort den Zweck des Actors erschließen.
- Trennzeichen zwischen Wortbestandteilen ist der Unterstrich „_“.
- Für die Namen von Actors innerhalb von Capsules immer die gleiche Bildungsvorschrift verwenden (z. B. Actors vom Typ **mod_engobj.loadtree** entsprechend immer „`loadtree`“, Actors vom Typ **mod_engobj.mode** immer „`mode`“ nennen etc.), das vereinfacht die Austauschbarkeit von Quelltext-Fragmenten und Java-Klassen.

Literaturverzeichnis

[Mic04] Sun Microsystems. „Code Conventions for the Java Programming Language“ Online-Dokumentation, Dezember 2004. http://java.sun.com/docs/codeconv/.