AniTMT–
Eine flexible Anwendung zum Erstellen fotorealistischer Filme und deren Berechnung in Netzwerken

von:
Jan Theofel,
Manuel Moser,
Martin Trautmann

Ein Beitrag aus dem Bereich Mathematik / Informatik
zum 35. Wettbewerb von Jugend forscht (2000).

Fassung für den
Landeswettbewerb Baden-Württemberg

9. März 2000

Inhaltsverzeichnis

1 Einleitung 2

2 Zielsetzung 3

3 Programmablauf 4

4 Anwendungsbeispiel 5
- 4.1 Szenenbeschreibung (POV-Datei) 5
- 4.2 Animationsbeschreibung (ADL-Datei) 6
- 4.3 Einstellungen (INI-Datei) 7

5 Technische Realisierung 8
- 5.1 Animationsberechnung (anitmt-calc) 8
  - 5.1.1 Variable Definition von Subfunktionen 8
  - 5.1.2 Wertebestimmung über Nachbarelemente 9
  - 5.1.3 Vergabe von Standardwerten 10
  - 5.1.4 Verweise auf Werte anderer Elemente 10
  - 5.1.5 Modularer Aufbau 10
- 5.2 Netzwerk 11
  - 5.2.1 Network render protocol (NRP) 11
  - 5.2.2 Serverprogramm (anitmt-server) 12
  - 5.2.3 Clientprogramm (anitmt-client) 12

6 Schlußbemerkung 12

Literatur 13

1 Einleitung

Filme wie *Toy Story* und *A Bug's Life* haben uns eindrucksvoll gezeigt, welche Qualität mit vollständig computer-generierten Filmen inzwischen erreicht werden kann. Die Erstellung solcher Filme – wenn auch in wesentlich bescheidenerer Form – ist für uns ein großer Traum seit wir uns mit Raytracing beschäftigen.

Für unsere ersten Versuche haben wir den Raytracer POV-Ray verwendet. Er hat uns im Bereich der Standbilder überzeugt, doch die darin integrierte Animationsfähigkeit ist in der Anwendung zu umständlich. Allerdings haben wir auch keine andere Software gefunden, die eine angemessene Lösung bietet.

Aus diesem Grund haben wir uns dazu entschlossen, ein eigenes Animationssystem zu entwickeln. Als Name hierfür haben wir *AniTMT* gewählt, wobei *TMT* für die Anfangsbuchstaben unserer Nachnamen steht.

Zunächst haben wir bestehende Lösungen dahingehend analysiert, warum sie für uns nicht in Frage kamen. Aus den so gefundenen Mängeln haben wir dann unsere Ziele wie folgt formuliert:

2 Zielsetzung

Das ganze System soll eine **mächtige Basis** zur Erstellung von Animationen darstellen. Um dies zu ermöglichen, entwickeln wir eine neue **Skriptsprache**, mit deren Hilfe selbst komplexe Animationen direkt erstellt werden können.

Diese kann später auch von anderen Programmen verwendet werden, was beispielsweise für die Entwicklung oder Anpassung von graphischen Modellierwerkzeugen hilfreich ist.

Daraus ergeben sich weitere Ziele:

- Teil der Mächtigkeit ist die **flexible Angabe von Parametern**, die voneinander abhängig sind. Die jeweils unbekannten werden automatisch ermittelt.
- Da die erforderliche Rechenleistung zur Erstellung von Animationen sehr hoch ist, haben wir deren **Verteilung im Netzwerk** als notwendigen Bestandteil in unser System integriert.
- Ein weiterer notwendiger Bestandteil ist die **Unabhängigkeit** vom Betriebssystem und von anderen Programmen.
- Damit das Animationssystem problemlos von anderen Anwendungen verwendet werden darf, wird es als **freie Software** verfügbar sein. Daraus erhoffen wir uns eine höchst effektive Zusammenarbeit, sowohl mit Anwendern als auch mit Programmierern, die eine kontinuierliche Verbesserung der Software ermöglicht.
- Weil auch wir „das Rad nicht neu erfinden“ möchten, verwenden wir zum Berechnen der Einzelbilder einen **externen Raytracer**. Hierbei haben wir uns vorerst für POV-Ray entschieden, da dieser sowohl unsere Ansprüche an eine Basissoftware erfüllt als auch die größtmögliche Anwenderschaft anspricht.
- Eine **modulare Implementierung** soll die Erweiterung um andere Raytracer ermöglichen und eine optimale Schnittstelle für externe Programme bereitstellen.

3 Programmablauf

Das folgende Schaubild zeigt den gesamten Ablauf innerhalb unseres Systems, das aus mehreren einzelnen Programmen besteht. Die einzelnen Schritte werden in den folgenden Abschnitten erklärt.

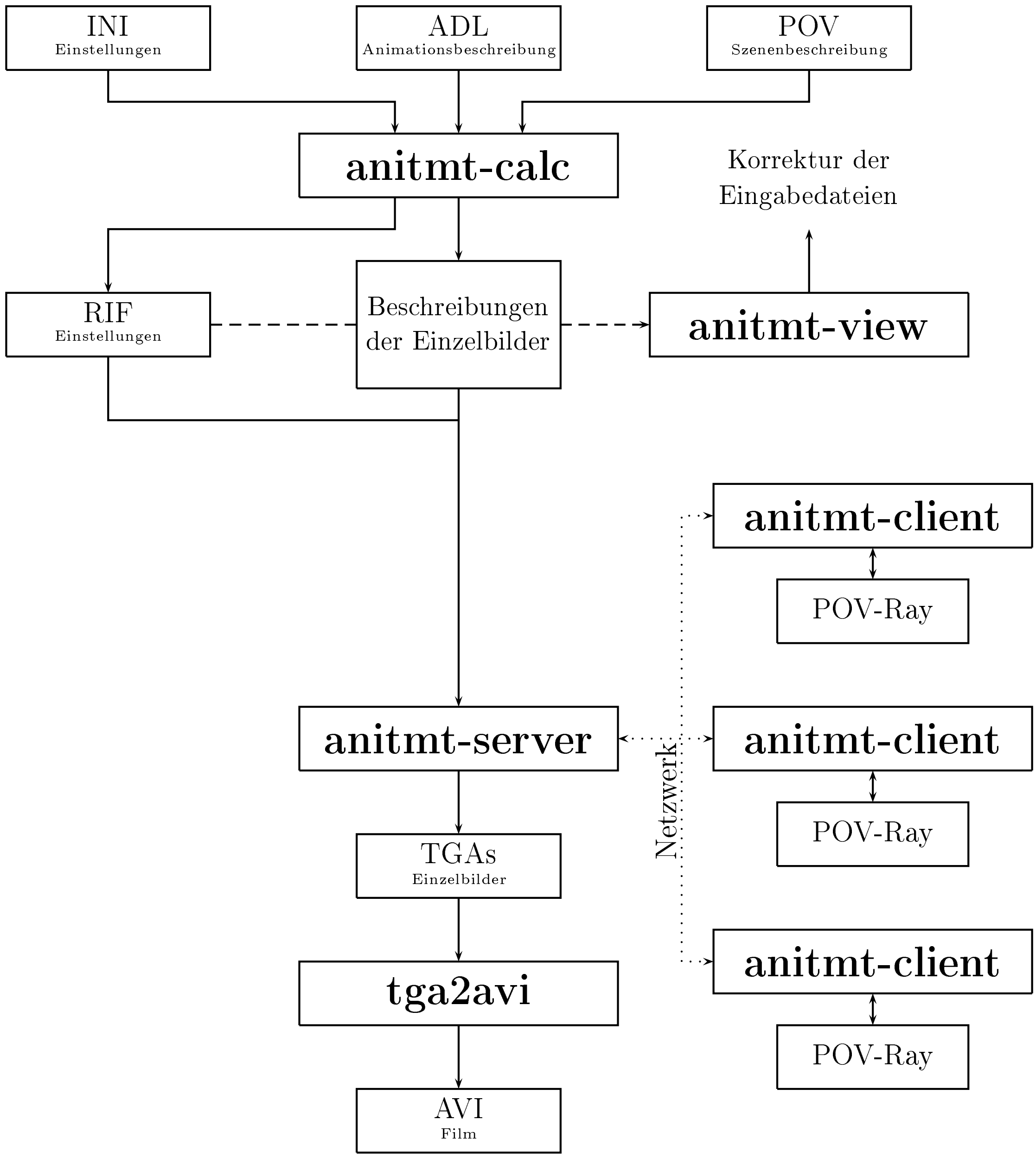

Abbildung 1: Übersicht des Programmsystems AniTMT

1. Zunächst erstellt der Anwender Szenenbeschreibungen für seinen Raytracer, wie zum Beispiel **POV-Ray-Dateien**, wobei die zu animierenden Komponenten benannt werden müssen. Durch diese geringe Änderung können bereits bestehende Standbilder zur Animation vorbereitet werden.

   In einer **ADL-Datei** wird der Ablauf der Animation festgelegt (siehe 4.2).

   Die Animationsparameter werden in einer **INI-Datei** (siehe 4.3) definiert und beim Aufruf an das Animationssystem übergeben.

2. Aus diesen Dateien berechnet das **lokale Animationssystem** `anitmt-calc` für jedes Einzelbild der Animation eine Szenenbeschreibung und ermittelt alle beteiligten Dateien.

3. Um die Animation zu testen, kann mit `anitmt-view` eine **Vorschau** berechnet werden. Hierzu wird eine geringe Auflösung und Bilderanzahl gewählt, sowie die rechenintensiven Effekte der Szene entfernt.

4. Ist die Animation fehlerfrei, werden die Dateien mit höheren Qualitätseinstellungen an das **Server-Programm** `anitmt-server` zur Verteilung im Netzwerk übergeben.

   Dort melden sich Rechner, die ihre Rechenzeit zur Verfügung stellen, mit dem **Client-Programm** `anitmt-client` an.

   Die Client-Programme liefern berechnete Bilder an den Server zurück, wo sie schließlich mit dem Programm `tga2avi` zu einem **Film** zusammenfügt werden.

4 Anwendungsbeispiel

Nachfolgend wird an dem Beispiel eines Kegels gezeigt, wie man mit unserem System eine Animation erstellt. Allgemein werden drei Eingabedateien[1] benötigt:

4.1 Szenenbeschreibung (POV-Datei)

```
cone{ // my_obj           <- Name des Kegels
  <-1, 0, 0>,                   // Fusspunkt
  0.5,                          // Radius am Fusspunkt
  < 1, 0, 0>,                   // Spitze
  0                             // Radius an der Spitze
  pigment{
    color rgb <1, 0, 0>         // Rot
  }
}

```

[1] Die Zeilennummern dienen nur zur Dokumentation und sind nicht in den Dateien enthalten.

```
11  camera{ // my_camera      <- Name der Kamera
12    location <0,1,8>              // Position
13    look_at <0,0,0>               // Betrachtungspunkt
14  }
15
16  light_source{
17    <1,1,5>                       // Position
18    color rgb <1,1,1>             // Weisses Licht
19  }
```

Zuerst wird eine Beschreibung der Szene benötigt. Als Beispiel haben wir einen Kegel definiert, dessen Mittelpunkt sich im Ursprung befindet und dessen Spitze in X-Richtung zeigt. Dies sind die Standardwerte unseres Systems für Drehzentrum und Vorne-Richtung.

Damit der Kegel sichtbar wird, ist zusätzlich eine Kamera zu positionieren (Die Kamera betrachtet den Ursprung aus der Z-Richtung.) und eine Lichtquelle zu definieren.

Als nächstes müssen alle Objekte, die animierbar sein sollen, benannt werden. Dazu haben wir die Syntax eines POV-Ray Modellers übernommen, der die Namen als Kommentar hinter der öffnenden geschweiften Klammer einfügt (Zeilen 1 und 11).

4.2 Animationsbeschreibung (ADL-Datei)

Nun wollen wir eine Flugbahn für den Kegel definieren. Dies geschieht mit Hilfe der von uns entwickelten *animation description language* (ADL).

```
povscene{
  filename "kegel.pov";

  my_obj {             // Name des Kegels
    move {               // Bewegung auf Flugbahn
      straight {           // Gerade zum Ursprung
        startpos    <-2,0,0>;       // Startposition
        startdir    x;              // Startrichtung
        length      3;              // Laenge in LE
        startspeed  2;              // Geschwindigkeit in LE/s
      }
      circle {             // 180 Grad Kurve
        normal      y;              // X-Z Ebene
        radius      2;              // Radius in LE
        angle       180;            // Winkel in Grad
      }
      circle {             // 130 Grad Kurve schief in den Raum
        center      <1,1,-1>;       // Rotationszentrum
        angle       130;            // Winkel in Grad
```

```
20        }
21        straight {}          // Gerade in die Unendlichkeit
22      }
23    }
24  }
```

Das Dateiformat besteht im Wesentlichen aus Blöcken und deren Eigenschaften. In einer hierarchischen Struktur wird zunächst die zu animierende Szene und dann die Komponente festgelegt, die sowohl ein Skalar, ein Vektor oder ein ganzes Objekt sein kann. Für die Komponente bestimmt eine Funktion die Art der Animation. Die Funktion `move` (Z. 5) bewegt ein Objekt durch Aneinanderreihungen von Bahnelementen. Die Elemente einer Funktion werden durch Subfunktionsblöcke definiert. In ihnen werden die Eigenschaften, die das eigentliche Verhalten der Animation bestimmen, festgelegt.

In diesem Beispiel bewegt sich der Kegel zunächst auf gerader Bahn in X-Richtung durch den Ursprung (Z. 6-11). Es folgt eine 180-Grad-Kurve in der X-Z-Ebene, definiert durch den Normalenvektor (Z. 12-16). Dann soll eine 130-Grad-Drehung um ein festes Rotationszentrum durchgeführt werden (Z. 17-20). Die abschließende Gerade ohne Eigenschaften sorgt für eine geradlinige Fortsetzung der Bahn bis zum Ende der Animation (Z. 21).

4.3 Einstellungen (INI-Datei)

```
[files]
adl=kegel.adl

[options]
fps=6
endtime=11
width=320
height=240
ani_dir=ani/
raytracer=povray3.1
params=+A0.3 -J
```

Im Abschnitt `[files]` werden Quelldateien der Animation angegeben. Die beteiligten Szenendateien werden in der Regel automatisch ermittelt, können jedoch auch explizit angegeben werden.

Im Abschnitt `[options]` werden Animationseinstellungen, wie zum Beispiel die Filmqualität, definiert. Im Normalfall werden die Anzahl der Einzelbilder pro Sekunde (`fps`), die Auflösung, die Dauer und der verwendete Raytracer angegeben.

Um die Berechnung zu starten, wird die INI-Datei an das Animationssystem übergeben.

5 Technische Realisierung

5.1 Animationsberechnung (anitmt-calc)

Für die Programmierung des lokalen Animationsprogramms haben wir uns für C++ entschieden, da diese Sprache für die Entwicklung von sehr großen Projekten geeignet ist. Außerdem sind diese Aufgaben mit Befehlen aus dem C++-Standard[2] realisierbar. Das Programm kann deshalb auf alle gängigen Betriebssysteme portiert werden. Bislang wurde es erfolgreich unter Linux und Windows getestet.

Zuerst werden die in der INI-Datei angegebenen Animationsbeschreibungen eingelesen und in eine interne Objekthierarchie umgewandelt. Die angegebenen Eigenschaften werden dabei von einem Parser interpretiert, der beliebige Verrechnung von Werten, Vektoren und Zeichenketten zulässt.

Danach sucht das Programm nach animierten Komponenten in den Szenenbeschreibungen und ermittelt weitere beteiligte Dateien (Texturen, Schriften, ...).

Als nächstes werden alle Unbekannten in der Objektstruktur ermittelt, die zur Berechnung der Szene für jedes Einzelbild nötig sind. Die Methoden dieser Ermittlung werden im folgenden beschrieben:

5.1.1 Variable Definition von Subfunktionen

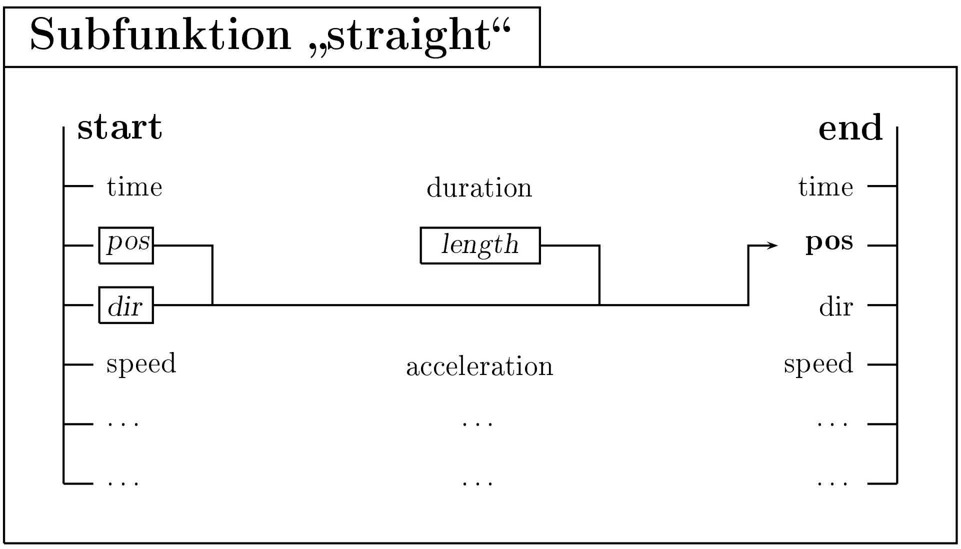

Abbildung 2: Variable Definition am Beispiel der Position

Jede Subfunktion besitzt verschiedene Eigenschaften, wie in Abbildung 2 dargestellt. Einige Beispiele: `starttime, endtime, startpos, endpos, duration,` ...

Wenn nun eine Eigenschaft benötigt wird, so sind alle Möglichkeiten bekannt, sie aus den anderen Eigenschaften zu berechnen. Bei einer geraden Bahn wird beispielsweise die

[2]ISO/IEC 14882, *Standard for the C++ Programming Language*

Endposition (**endpos**) aus Startposition (*startpos*), Startrichtung (*startdir*) und Länge (*length*) berechnet. Falls diese Eigenschaften ebenfalls nicht angegeben wurden, bestehen wiederum mehrere Möglichkeiten, auch diese Werte zu ermitteln.

Für dieses Fragesystem haben wir Methoden entwickelt, um unnötige Anfragen zu vermeiden.

5.1.2 Wertebestimmung über Nachbarelemente

Die Ermittlung der benötigten Eigenschaften erfolgt in mehreren Stufen. Zuerst werden in jedem Abschnitt die Unbekannteni, wie gerade beschrieben, ermittelt. Wenn Subfunktionen nicht sämtliche ihrer Eigenschaften berechnen konnten, werden entsprechende Werte von Nachbarelementen übernommen.

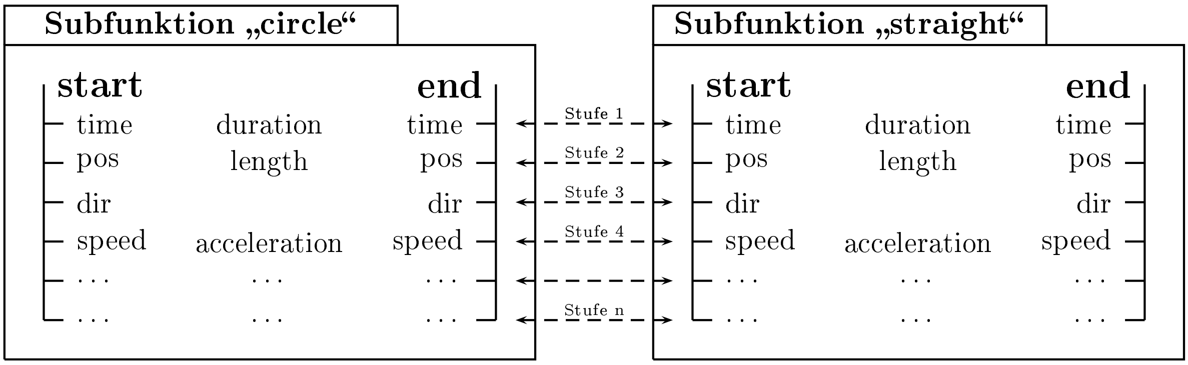

Abbildung 3: Wertebestimmung über Nachbarelemente

Um zum Beispiel bei einer Flugbahn einen zeitlich zusammenhängenden Vorgang sicherzustellen, sollten die Start- und Endzeiten benachbarter Elemente übereinstimmen. Um Sprünge zu vermeiden, benötigt man gleiche Positionen; um Knicke zu verhindern, gleiche Richtungen; um die Animation flüssig zu gestalten, gleiche Geschwindigkeiten. Auf jeder Stufe wird eine Eigenschaft in obiger Reihenfolge zur Übergabe freigegeben.

Damit ist es möglich, eine Animation mit relativen Veränderungen, wie der Dauer, zu definieren und nur an einer Stelle absolute Werte, wie zum Beispiel die Startzeit, anzugeben.

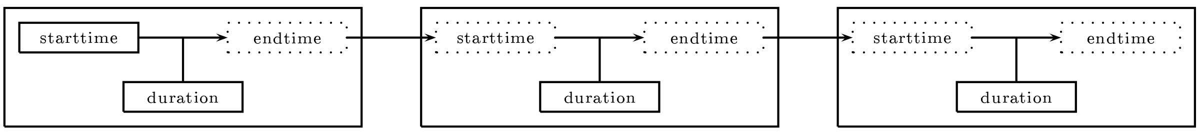

Abbildung 4: Verkettung ohne absolute Werte

5.1.3 Vergabe von Standardwerten

Da wir sowohl mächtige als auch komfortable Funktionen anbieten wollen, werden bei Abschnitten, die noch nicht eindeutig definiert sind, Standardwerte gesetzt. Damit wird beispielsweise statt einer beschleunigten Bewegung eine gleichförmige Bewegung angenommen. Einige Anfangswerte des ersten Abschnitts werden standardmäßig gesetzt. Auch die Endzeit des letzten Abschnittes kann mit Hilfe der Endzeit der gesamten Animation ermittelt werden.

5.1.4 Verweise auf Werte anderer Elemente

Desweiteren ist es möglich, Beziehungen zwischen mehreren Bewegungen herzustellen.

Beispielsweise kann man eine Rakete durch Verweise auf die Eigenschaften eines Flugzeuges so ausrichten, daß sie dieses zum richtigen Zeitpunkt trifft, ohne die exakte Position und Zeit zu bestimmen.

Generell ist es auch möglich, die Werte weiter zu verrechnen, um beispielsweise die Rakete das Flugzeug verfehlen zu lassen, indem sie die Bahn erst einige Sekunden später kreuzt.

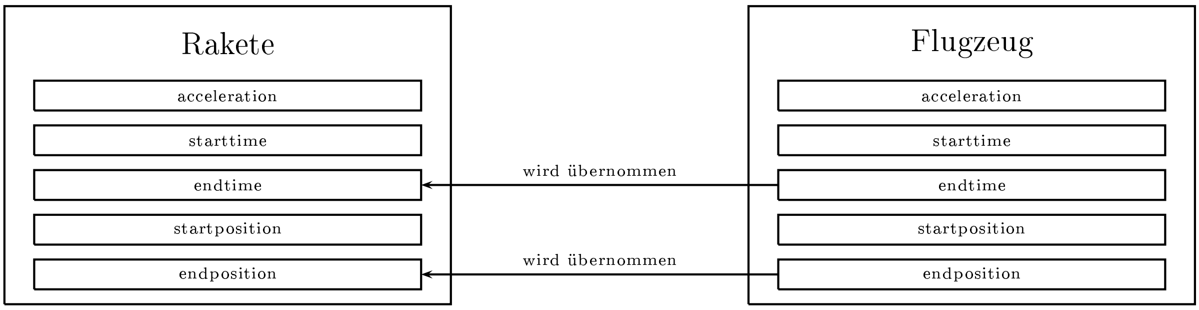

Abbildung 5: Verknüpfung zwischen beliebigen Elementen

5.1.5 Modularer Aufbau

Bei der Programmierung wurde besonders Wert auf Modularisierung gelegt. Dies ermöglicht sowohl die Unterstützung von weiteren Raytracern, als auch neue Formate zur Animationsbeschreibung. Besonders wichtig ist auch die Möglichkeit, das System relativ einfach um neue Animationstypen erweitern zu können.

Bisher haben wir exemplarisch die wichtigsten Funktionen implementiert. Dazu gehört die lineare und quadratische Interpolation von Werten, die in POV-Ray sehr mächtig eingesetzt werden können (z. B. zur Drehung von Gelenken in Robotern).

Weiterhin können ganze Objekte, wie beispielsweise Raumschiffe, auf Flugbahnen aus Geraden und Kreiskurven bewegt werden. Sowohl die Bewegung auf der Bahn als auch eine beliebige Rotation werden einheitlich von einem bahnunabhängigen Grundtyp bereitgestellt. Auf diesem können neue Bahntypen aufbauen, wie beispielsweise Bezierkurven, an deren Implementierung wir gerade arbeiten.

Auch neue Animationsarten, wie Vektoranimation, Bewegung durch Kraftwirkung oder eine spezielle Rotationsfunktion, können problemlos hinzugefügt werden. Derzeit arbeiten wir an einem komplexen Animationstyp für die Bewegung von Objekten, der in drei Bestandteile aufgeteilt ist: Die Flugbahn, die Blickrichtung und die Rotation um diese.

Der Vorteil dieses Systens ist, daß für jeden der Bestandteile verschiedene Funktionen beliebig miteinander kombiniert werden können. Andererseits können die Bestandteile auch völlig frei voneinander definiert werden, was zu größtmöglicher Flexibiliät führt.

5.2 Netzwerk

Zur Netzwerkimplementierung haben wir Perl verwendet, weil sich damit eine textbasierte Netzwerkkommunikation sehr leicht realisieren läßt. Außerdem ist es für alle gängigen Betriebssysteme verfügbar. Der durch diese Skriptsprache entstehende Geschwindigkeitsverlust fällt gegenüber dem enormen Rechenaufwand beim Raytracing kaum ins Gewicht.

Zur Kommunikation haben wir das *network render protocol* (NRP) entwickelt, das auf TCP/IP als dem gängigsten Netzwerkstandard aufbaut. Hierdurch ist es theoretisch auch möglich, die Rechenlast über das Internet zu verteilen, was aber aus Sicherheitsgründen momentan noch nicht eingesetzt werden sollte.

Auf den Einsatz bestehender Clustering oder Verteilungslösungen haben wir bewußt verzichtet, da diese meistens an bestimmte Systeme gebunden sind. Außerdem müssten bei ihrer Anwendung weitere externe Programme konfiguriert werden.

`anitmt-server` bezeichnet das Programm, das auf dem Server[3] läuft. Auf den Clients wird als Gegenstück `anitmt-client` gestartet.

5.2.1 Network render protocol (NRP)

Bei der ersten Anmeldung überträgt der Client einige Informationen, wie beispielsweise Geschwindigkeit und installierte Raytracer. Diese werden mit der anschließend zugewiesenen Client-ID (CID) verküpft, so daß sie nur einmalig übertragen werden müssen.

Das Protokoll ist in verschiedene Modi eingeteilt, wobei es neben dem normalen Rechenmodus auch noch einen zum Debuggen des Servers und einen zum Verwalten der Projekte gibt. Momentan gibt es noch keinen extra Modus zur Datenübermittlung. Statt dessen wird auf ein Netzwerkdateisystem zurückgegriffen.

Alle Befehle, die an den Server geschickt werden, sind vier Buchstaben lange Abkürzungen. Bei den Antworten wird jeder Zeile ein dreistelliger Zahlencode vorangestellt. Die erste dieser Ziffern gibt den Protokollabschnitt an, die zweite informiert über möglicherweise aufgetretene Fehler und die dritte regelt, ob der Server noch weitere Zeilen sendet oder der Client seinen nächsten Befehl schicken kann.

[3] Im folgenden wird der Rechner, der die Rechenlast verteilt, als *Server* bezeichnet und die Rechner, welche die eigentliche Berechnung durchführen, als *Clients*.

Momentan wird das Protokoll um die Unterstützung von FTP und einem eigenen internen Übertragungsmodus ergänzt, so daß es fast keine Softwarevoraussetzungen an die verwendeten Systeme mehr stellt.

5.2.2 Serverprogramm (anitmt-server)

Nach dem Starten des Servers überwacht das Serverprogramm den ihm zugewiesenen Port (Standard 4004) und startet für jede eingehende Verbindung einen Child-Prozess. Die dabei nötige Kommunikation zwischen den einzelnen Prozessen geschieht mit Hilfe von gemeinsam genutztem Speicher nach System-V-Standard. Dadurch scheiden einige Betriebssysteme als Serversystem leider aus.

Um gegen Serverausfälle gesichert zu sein, werden alle Informationen über die Clients zusätzlich in einer Datei gespeichert. Die noch zu berechnenden Einzelbilder einer Animation können aus den lokal vorhandenen Dateien rekonstruiert werden, sofern dies notwendig sein sollte.

Aktuell wird noch die Funktion, mehrere Projekte gleichzeitig mit verschiedenen Prioritäten zu verwalten, implementiert. Diese können ebenfalls über das Netzwerk verwaltet und überwacht werden. Daneben wird der Server zum echten *deamon* ausgebaut, der sofort mit dem Rechner gestartet wird und im Hintergrund läuft.

Weiterhin wird die Implementierung von redundanten Servern und ganzen Serverhierarchien angestrebt. Auch eine bessere Anpassung an stabile (LAN) und instabile (WAN) Netze ist in Arbeit.

5.2.3 Clientprogramm (anitmt-client)

Auf der Client-Seite ist das Programm deutlich einfacher aufgebaut. Das Programm versucht eine Verbindung zum Server aufzubauen und fordert dabei eine Rechenaufgabe an. Kann der Server nicht erreicht werden oder sind keine Aufgaben vorhanden, so wird der Versuch später wiederholt.

Wird dem Client eine Aufgabe zugeteilt, so ermittelt dieser die hierzu noch benötigten Dateien und fordert diese vom Server an. Nach deren Übertragung meldet er sich ab, startet den Raytracer zur Berechnung und baut zur Ergebnisübermittlung wieder eine Verbindung auf. Danach wird eine weitere Aufgabe angefordert.

Der Client wurde bislang nur unter Linux getestet, wird aber momentan auf andere Systeme portiert.

6 Schlußbemerkung

Obwohl wir noch sehr viel Arbeit vor uns haben, um alle unsere Ziele und Ideen zu verwirklichen, haben wir schon jetzt ein ausgereiftes System.

Es ist sehr einfach zu bedienen, bietet mächtige Funktionen, kann flexibel erweitert werden und ist netzwerkfähig. Wie wir mit unseren Beispielen zeigen können, ist AniTMT bereits produktiv einsetzbar.

Die Programme und zugehörigen Dokumentationen (Literaturangaben [5] bis [8]) sind unter der URL

`http://www.theofel.de/anitmt/`

zu finden. Dort gibt es auch Demos, darunter das oben besprochene Beispiel des fliegenden Kegels.

Danksagung

Folgenden Personen möchten wir unseren Dank aussprechen:

- Unseren Eltern, die uns den Rücken freigehalten haben, so daß wir uns immer ganz auf unsere Arbeit konzentrieren konnten.
- Dem Stuttgarter TeX-Stammtisch für das Korrekturlesen und die wertvollen Tips.
- Der gesamten Hacker-Gemeinde, die durch die Programmierung unzähliger kleiner und großer Programme die Umgebung für unser System geschaffen hat.
- Allen anderen Personen, die direkt oder indirekt zu dieser Arbeit beigetragen haben.
- Und natürlich allen, die den Wettbewerb Jugend-forscht erst ermöglicht haben.

Literatur

[1] Martin Brown. *Perl Annotated Archives.* Osborne, Berkley, 1999.

[2] Tom Christiansen and Nathan Torkington. *Perl Kochbuch.* O'Reilly & Associates, Inc., Köln, 1999.

[3] POV-Team. *Persistence of Vision Ray-Tracer – User's documentation 3.0.10*, 1996.

[4] Bjarne Stroustrup. *Die C++ Programmiersprache.* Addison–Wesley, 1997.