Visualisierungs-Frontend für Fahrzeugsimulatoren

Heinrich P. Godbersen

Vortrag: Simulation und Test. 3. Tagung 26-27 Mai 2008, Berlin.
Erschienen in: Gühmann, C. (Hrsg.): Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik II, Expert Verlag, ISBN: 978-3-8169-2818-8, Mai 2008, p. 188-194

Abstract

A new low-cost approach for the programming of real-time visualizations is introduced, based on the authoring system Quest3D. Some aspects concerning complexity handling in such an environment are discussed. A prototype implementation and its binding to an industrial vehicle simulation system is explained.

Kurzfassung

Ein neuer Low-Cost Zugang zur Programmierung von Echtzeitvisualisierungen auf der Basis des Autorensystems Quest3D wird beschrieben. Einige Aspekte der Komplexitätsbewältigung werden diskutiert. Die prototypische Anbindung an ein industrielles Fahrzeug- Simulationssystem wird vorgestellt

1. Einleitung

An der TFH Berlin wird seit Jahren in Rahmen anwendungsorientierter Forschung an neuen Methoden zur Echtzeitvisualisierung gearbeitet. Ein zentrales Ziel dabei ist, die Kosten solcher Anlagen deutlich zu reduzieren. Dazu finden Bausteintechniken sowohl bei der Hard- als auch der Software ihren Einsatz [Go07].

Als Softwarebasis wird das 3D-Autorensystem Quest3D eingesetzt. Quest3D unterstützt die datenflussorientierte graphische Programmierung von Echtzeitanwendungen auf der Basis von DirectX. Unsere Applikation nutzten zur Ermittlung der Fahrdynamik die eingebauten Physics-Engines ODE bzw. Newton.

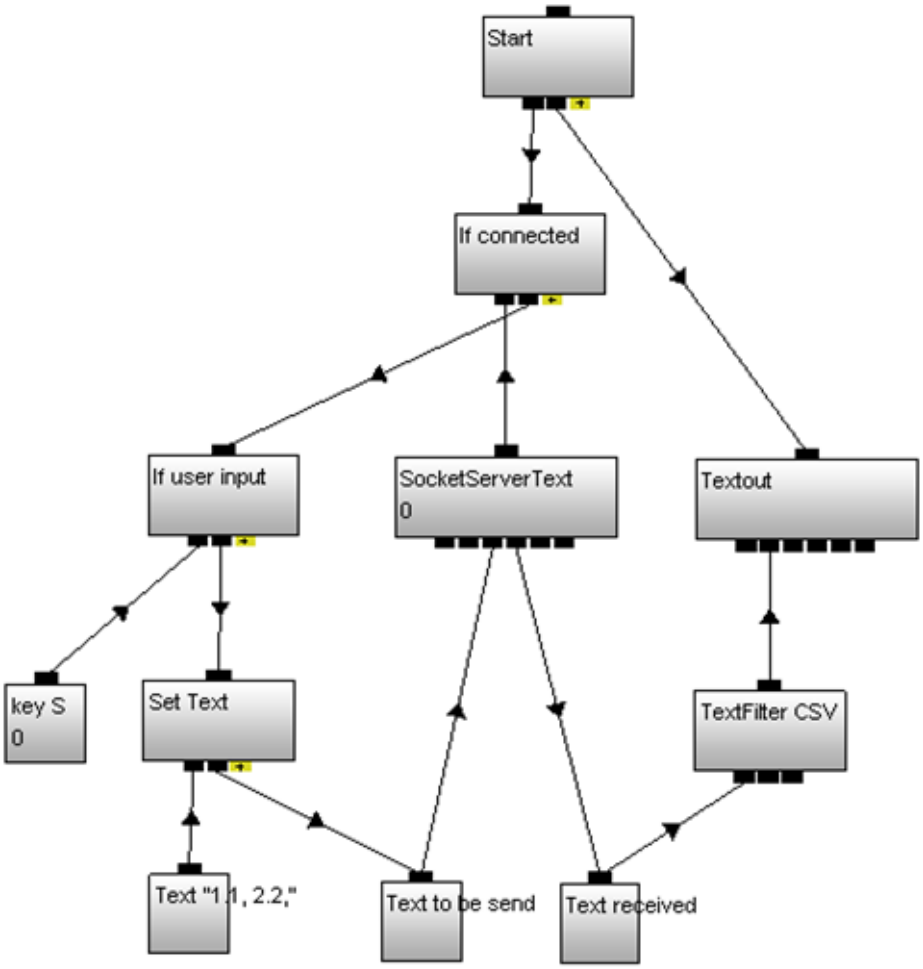
In Zusammenarbeit mit dem Unternehmen IAV wurde in Rahmen einer Diplomarbeit untersucht, wie sich Low-Cost Echtzeitvisualisierungen von Fahrzeugen auf der Basis exakterer Dynamikmodelle realisieren lassen. Dabei kommt deren eigenes Programmpaket VeLoDyn. (Vehicle Longitudinal Dynamics Simulation) zum Einsatz. Als Entwicklungssystem für die Visualisierung wurde wiederum Quest3D ausgewählt.

Ziel dabei war, ein eigenständiges Visualisierungs-Frontend zu bauen. Dazu musste u.a. eine Echtzeit-Kommunikation zwischen der Simulations-Engine Simulink und dem neuen Frontend aufgebaut werden.

2. Programmgenerierung für Echtzeitsysteme

Zur Softwareerstellung von Multimedianwendungen kommt bei uns das Autorensystem Quest3D auf breiter Basis zum Einsatz. Quest3D basiert auf dem Paradigma der datenflussorientierten graphischen Programmierung (s. [Go82], [Go82a], [Go92]): Aus einem Pool von rund 300 vorbereiteten Bausteinen wird ein Traversierungsbaum (scene graph) zusammengestellt. In dem Graphen sind letztlich alle Algorithmen untergebracht. In einem Publizierungsdurchlauf wird daraus ein Exe oder eine Web-Applikation erzeugt (s. [Pi04], [Go04b], [Go07a]).

Das folgende Beispiel zeigt einige wichtige Konzepte am Beispiel der Serverseite einer TCP Verbindung:



Ausgehend von der Baumwurzel wird zunächst die erste if- Abfrage evaluiert. Dazu wird zunächst das erste Kind SocketServerText aufgerufen. Dieser Channel realisiert einen TCP Server für Textaustausch. Falls bereits eine Client-Verbindung existiert, wird untersucht, ob bei dem ersten Kindknoten Daten zur Übertragung anliegen. Falls vorhanden, wird jetzt jeweils das Senden bzw. der Empfang durchgeführt. SocketServerText liefert als Ergebnis zum Elternknoten den Wert 1(= connected) ab. Dies führt bei der Evaluation von „if connected“ jetzt zum Aufruf von „if-user input“, der dann prüft, ob die Taste „S“ gedrückt ist. Im Erfolgsfall wird der vorbereitete Text „1.1, 2.2,“ in den Knoten „Text to be send“ kopiert. Die Evaluation des rechten Teilbaums führt – falls vorhanden - zur Extraktion des ersten empfangenen CSV-Wertes und seiner anschließenden Ausgabe auf dem Bildschirm. Dieser Zyklus wird ständig durchlaufen.

Bild 1: TCP Server Betrieb unter Quest3D

Die vorhandenen Bausteine decken neben dem Zugang zu DirectX anwendungsbezogene Algorithmen zur Physik und Bewegungsplanung ab. Mit dieser Technik wurde von uns u.a. das Rennspiel „Duo 3000“ produziert [Be05]:

Bild 2: Rennspiel „Duo 3000“ in der Multiscreenumgebung der TFH Berlin

2.1. Kapselung

Datenflussorientierte Programmieransätze sind auch in hardwarenahen Umgebungen vielfältig in Gebrauch, z.B. bei Simulink und LabVIEW. Der graphische Programmieransatz von Quest3D baut auf dem in der Computergrafik üblichen Konzept des Scenegraphs auf[1]. Der Baum wird typischerweise zyklisch in Postorder traversiert, dabei kommen die jeweiligen Algorithmen in den Knoten zur Ausführung.

Die Überprüfung der Datenstruktur-Konsistenz geschieht bereits im WYSIWYG Editor. Das folgende Beispiel zeigt die Kapselung der typischen Bibliotheksfunktion atoi() „ASCII to integer“ in einem Quest3D channel:

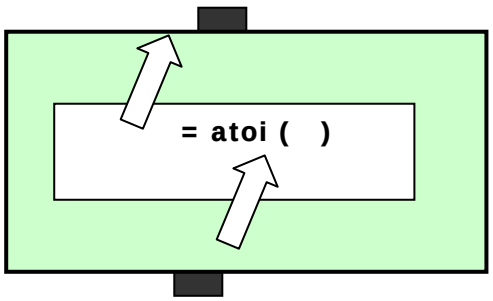


Bild 3: Kapselung einer Funktion

[1] Wir verwenden den Begriff Baum hier übergreifend. Quest3D setzt einen gerichteten Graphen ein, der wegen Schreib-Operationen sogar Zyklen enthalten kann, die durch eingebaute Mechanismen gesteuert werden.

Diese Technik der Kapselung ist für alle definierten Datenstrukturen im System (einschließlich unions) gleich, sie kostet nur einen geringen Overhead. Die exemplarische Verwendung von atoi in einem Scengraph zeigt die folgende Abbildung:

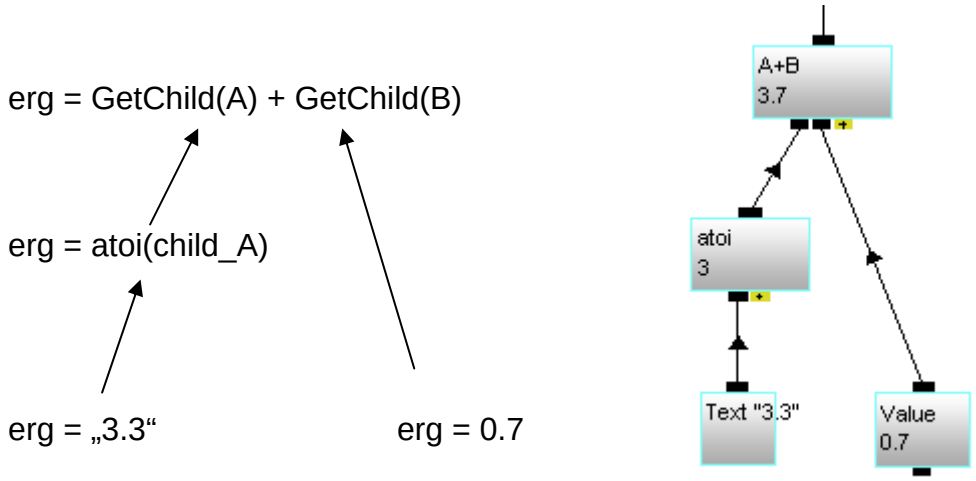


Bild 4: Abarbeitung eines Graphen

2.2. Handhabung komplexer Projekte

Bei solcherart kleinen Funktionen spricht der Programmieraufwand eher zum Nachteil von Quest3D. Es gilt daher, möglichst große wieder verwendbare Einheiten zur Verfügung zu stellen. Die folgende Abbildung gibt die Größe der jeweiligen DLLs für die ca. 320 Bausteine (Channels) an. Angaben in KByte, Quest3D Version 4.0:

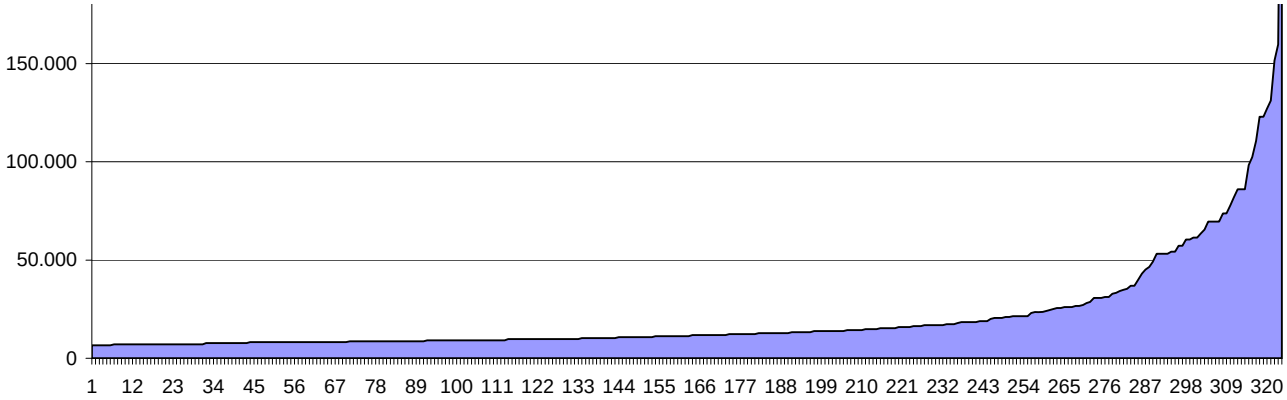


Bild 5: Bausteingröße, sortiert

Ein Baustein wie atoi würde ganz links angesiedelt sein. Der oben erwähnte Channel SocketServerText hat dagegen einen Footprint von ca. 60 KB. Alle Bausteine benötigen ca. 5KB für die Kapselung selbst.

Die Effizienz bei der Produktion ist am größten, wenn die Bausteine möglichst viele Standardsituationen umfassend abbilden. Dies ist für den Bereich des Geometrieimports und der Direct3D-Darstellungspipeline gut gesichert.

Da aber auch die individuelle Programmlogik im Baum unterzubringen ist, droht eine Explosion der Komplexität der Baumstruktur. Hier gilt es, geeignete Strukturen zur Hierarchiebildung bereitzuhalten. Dabei bieten sich u.a. folgende Strategien an:

- Bereitstellung von wieder verwendbaren Templates
- Vergröberung eines Teilbaums zu einem Knoten
- Bündelung der Entscheidungsfindung in Zustandsmaschinen (FSM)
- Auslagerung von Algorithmen in eine Skriptsprache
- Neuerstellung eines Bausteins mittels SDK
- Konzepte der OO-Programmierung verwenden

3. Lösungsansatz

Es sind drei Aspekte zu nennen:

3.1. Kommunikationsinfrastruktur

Hier war es u.a. aus lizenzrechtlichen Gründen erforderlich, für Quest3D einen neuen Channel zu programmieren. Dessen Weiterentwicklung wurde bereits oben als SocketServerText vorgestellt.

3.2. CSV Datensätze

Eine spezielle Aufgabe bestand darin, Simulationsergebnissen in Echtzeit zwischen Systemen übertragen. Als robuste Lösung wurde ein CSV-Ansatz eingesetzt. Die ausgetauschten Datensätze enthalten ausschließlich ASCII-Text. In unserem konkreten Fall werden pro Simulationsschritt zehn Vektoren übertragen, die die Position und Drehung von Chassis und allen vier Rädern des Fahrzeuges definieren. Als Trennoperator für diese 30 aufgereihten Real-Zahlen kommt das klassische Komma zum Einsatz (Channel: TextFilter).

3.3. Synchronisation der beteiligten Prozesse

Auf der Visualisierungsseite ist die aktuelle Framerate nicht exakt vorgebbar, denn sie ist abhängig von Hardwaregegebenheiten und der aktuellen CPU-Last. Mehrere Strategien stehen zur Auswahl:

1. Schritthaltende Verarbeitung: De Verarbeitungsfrequenzen von Erzeuger- und Konsument stimmen weitgehend überein.
   - Der jeweils aktuellste zur Verfügung stehende Simulationsschritt wird visualisiert. Dabei kann es Auslassungen und Jitter kommen. Im Idealfall ist das nicht visuell zu erkennen. Eine explizite Rückkopplung ist nicht nötig. Diese lose Kopplung ist am einfachsten zu realisieren und vielfältig im Einsatz.
   - starke Kopplung: Hier wird dem Erzeugerprozess eine Rückkopplung gegeben.

2. Große Differenzen in den Geschwindigkeiten, führen zur einer Batchverarbeitung bzw. Offline-Simulation.
   - Zwischenspeicherung der Simulationswerte auf der Produktionsseite (z.B. als Datei)
   - Interpolation der Simulationswerte durch Splines oder Vektorinterpolation. Hiermit wäre auch eine Visualisierung in Slow-Motion möglich.

Für alle genannten Varianten stehen in Quest3D Bordmittel zur Verfügung.

4. Prototyp AmiCas

In der Diplomarbeit von Frau Varavina [Va07] wurde unter Quest3D der Prototyp eines Frontends unter dem Namen „AmiCas“ entwickelt. Dabei musste auch auf der Seite der Simulink-Applikation die Programmierung eines TCP Clients realisiert werden. Eine Vorgängerversion des oben erwähnten „SocketServerText“ wurde mit dem SDK unter C++ geschrieben.

Der Datenaustausch geschieht mit CSV über eine TCP-Verbindung, bzw. Offline aus einem Textfile heraus. Die übertragenen Messwerte werden auch in einem TCB-Spline interpoliert bereitgestellt. Es lassen sich unterschiedliche Fahrzeugmodelle aus einem Pool laden:

Bild 6: Screenshot AmiCas

Der Prototyp wurde erfolgreich getestet. Dieser verteilte Lösungsansatz erlaubt eine effektive Produktion ergänzender Software.

5. Ausblick

In einer Fortschreibung des Projekts wird die alternative bzw. ergänzende Verwendung von weiteren Physik-Engines betrachtet. Ferner bietet sich auch der Ausbau zu einem Interaktions-Frontend an, das sowohl für Testzwecke als auch für Kunden-Präsentationen verwendet werden kann.

Es stehen neue Bausteine im Bereich GUI-Erstellung und dem Geometrie-Import mittels Collada bereit. Darüber hinaus stehen inzwischen auch attraktive Ansätze von Generatoren zur Shaderprogrammierung zur Verfügung. Damit könnte die Produktivität solch eines Vorgehens weiter gesteigert werden.

Letztendlich steht die Vision eines zukünftigen Zusammenwachsens der Herangehensweise bei Spieleprogramierung und industriellen VR Systemen im Raum [Th07], [Go08].

Literatur

[Be05] Becker, M.; Lehmann, J-F.; Ringel, O.; Schwarz, S.: DUO 3000. Interaktives 3D Spiel. Projektbericht, TFH Berlin 2005, Betreuerin: Prof. M. Kothe

[Go83] Godbersen, H.P.: Funktionsnetze. Eine Modellierungskonzeption zur Entwurfs- und Entscheidungsunterstützung. Ladewig Verlag, Birkach, Berlin, München, 1983

[Go83a] Godbersen, H.P.: Simulation with "FUN". Angewandte Informatik, 5/83, pp. 213-21

[Go92] Godbersen, H.P.; Kroll, A.; Pfafferott, A.: Netzbasierte Software-Synthese verteilter Anwendungen. Proc. Softwaretechnik in Automation und Kommunikation (STAK '92), VDI Berichte Bd. 937, 1992, p. 199-210

[Go04b] Godbersen, H.P.: Erstellung von Anwendungen für eine Virtuelle Umgebung. TFH Berlin, Interner Bericht, Q1, 2004

[Go07] Godbersen, H.P.: Eine Low-Cost Lösung für interaktive, immersive 3D-Echtzeitanwendungen. erschienen in: Paul, L.; Stanke, G.; Pochanke, M.: 3D-NordOst 2007, GFaI Berlin, Dezember 6-7, 2007. ISBN: 3-9809212-9-8, p. 131-138

[Go07a] Godbersen, H.P.: Teaching Computer Graphics and Multimedia with Quest3D. Second Quest3D Conference, Aachen, Sept. 13-14, 12007

[Go08] Godbersen, H.P.: Virtual Environments for Anyone. *Zur Veröffentlichung angenommen.*

[Pi04] Pieper, J.; Godbersen, H.P.: Echtzeit-3D-Visualisierung. immersiv und verteilt. TFH Berlin, Interner Bericht, Q1, 2004

[Th07] Thadeusz, F.: Sturzflug für jedermann. Der Spiegel, 27/2007, S. 151

[Va07] Varavina, M.: Erstellung einer Animationsschnittstelle für das Fahrzeugsimulationsprogramm VeLoDyn. Diplomarbeit, TFH Berlin, 2007