Jagdszene 0.1 alpha

von Michael Härdi, Dezember 2007

Dieser Bericht beschreibt den Versuch, unter ausschliesslicher Verwendung von freier Software ein Spiel zu programmieren. Um es gleich vorweg zu nehmen – ein Spiel ist es noch nicht geworden, aber immerhin eine animierte Figur, welche mit den Pfeiltasten bewegt werden kann. Die Abfolge der Ereignisse ist in chronologischer Form und die einzelnen Versuche geschehen nicht zwingend systematisch. Die Einträge erfolgen als "Tagesberichte" – die Beschreibungen zu den einzelnen Software-Paketen sind nicht immer zusammenhängend. Die Motivation für diese Arbeit ist es, gewisse "Geheimnisse" zu lüften, welche mich als Spieler von Computerspielen schon immer interessiert haben.
Ich beschreibe gleichzeitig eine Gemeinschaft von Personen, die sich mit der Entwicklung von Spielen beschäftigt und zudem ihre Resultate auf dem Internet veröffentlicht. Diese Gruppe ist eher klein, weil es sich um ein komplexes Thema handelt, welches ein grösseres Vorwissen erfordert, um überhaupt nachvollziehbar zu sein. Oft stammen die Entwickler von Open Source Software aus dem Umfeld von technischen Universitäten.
Ich versuche, den Text so zu verfassen, dass er "normal" lesbar bleibt – die kryptischen Stellen können auch überlesen werden.
Beim Nachforschen und Evaluieren habe ich mich relativ konsequent an die Regel gehalten, nur Software zu verwenden, welche unter der uneingeschränkten GPL-Lizenz [1] verfügbar ist. Weiter habe ich versucht, alle diese Programme als Source-Code herunterzuladen und selbst zu kompilieren (Mit "Kompilieren" ist, kurz gesagt, eine Art "Zusammenleimen" von vielen einzelnen Dateien zu einem Programm gemeint). Dadurch verfüge ich nun über eine "Programmier-Umgebung", in der alle Dateien "offen" in den Programmiersprachen C++ [2] oder Python [3] vorhanden sind.

1. Tag

Die Jagd nach der Software beginnt: Ich tippe in die Suchmaschine [4] die Worte "authoring software" ein. Ziemlich bald komme ich zur Seite der 3D-Software Blender [5]. Es handelt sich dabei um ein Programm, welches mit Open Source Software entwickelt wurde. In einem Bericht des Grafik-Chip-Herstellers Intel [6] wird die Rendering-Engine Irrlicht [7] empfohlen. Ich lade sie auf SourceForge [8] herunter. Dabei erfahre ich, dass SourceForge ein wichtiger Ort ist für Open Source Projekte. Eine grosse Anzahl davon steht hier zum Download bereit und zu vielen davon finden sich auch Beschreibungen und Installationsanleitungen. SourceForge ist vorallem eine Austauschplattform für Software-Entwickler – es würde also die Möglichkeit geben, Projekte herunterzuladen, welche mitten in der Entwicklung stehen. Ich denke aber, es ist eine gute Idee, sich an die "letzte stabile" Version zu halten, denn auch damit gibt es meistens noch genug Probleme zu bewältigen, um sie zu kompilieren und zu installieren. Zu einigen Projekten gibt es für Windows fertige installierbare Programme.

Bei Microsoft [9] beschaffe ich mir die Gratis-Software Visual C++ Express Edition. Ich lasse die C-Sharp und die .NET Version weg, da ich vorhabe, ausschliesslich C++ Software zu kompilieren. Viele Open Source Projekte sind ursprünglich im Betriebssytem Unix oder Linux entwickelt worden und liegen nur in C++ Code vor. Auch ist es so möglich, die Projekte zu einem späteren Zeitpunkt auf Linux oder Apple OSX zu kompilieren. Zum Einstieg in die Thematik werde ich vorläufig nur im Betriebssytem Windows arbeiten.

2. Tag
Ich mache mich mit Blender vertraut. Vorher installiere ich Python, da Blender intern mit Python-Scripten arbeitet. Es gibt auf dem Netz viele gute Anleitungen und Tutorials – ich versuche, gleich mal die "Introduction to the Game Engine" durchzuarbeiten. Es gelingt mir sehr schnell, eine Kugel ins Rollen zu versetzen oder einen Würfel zu verschieben. Ich mache von den Runtime-Saves (auch eine Art von Kompilieren) Kopien auf den USB-Stick und teste die Beispiele auf einem zweiten Windows-Rechner. Damit sie funktionieren, muss ich noch einige dll-Dateien mit in den Ordner kopieren. Warum und welche wird in der Anleitung erklärt; jedenfalls ist es nicht so schwer und schliesslich laufen die Beispiele.

3. Tag
Heute schaue ich eine Einführung in die Programmiersprache Python an. In Blender sind viele Funktionen schon vordefiniert und es können recht intuitiv Verknüpfungen zwischen diesen Objekten erstellt werden. Es tut gut, einmal eine fundierte Einführung zu lesen über die Grundlagen des dreidimensionalen Arbeitens. Ein dreidimensionales Gitter wird Mesh genannt, der einzelne Punkt Vertex, in der Pluralform Vertices. Dann haben wir die Linien bzw. Edges und die Flächen bzw. Faces. Ein Dreieck besteht demnach aus 3 Vertices, 3 Edges und einer Fläche. Das ist reine Vektorgeometrie. Um mit der Syntax etwas anfangen zu können, muss ich nun erfahren, wie die Objekte adressiert sind bzw. wie sie angesprochen werden können.

4. Tag
Eine Szene ist eine Gruppe von Objekten in einer dreidimensionalen Landschaft. Dazu gehören auch die Beleuchtung (Sonne oder Lampen) und die Kamera, aus deren Sicht der Betrachter die Szene schliesslich sehen wird. Ich mache eine Python-Übung durch, in der die Koordinaten aller dieser Objekte erfasst werden und in eine Text-Datei geschrieben werden. Dann folgt ein Beispiel, wo diese Text-Datei in eine leere Szene importiert wird. Ich bemerke, dass nur das erste Objekt importiert wird. Also schreibe ich das Script etwas um, kurz gesagt ein Array und eine Schlaufe, bis dass alle Objekte importiert werden. Das Exportieren und Importieren von Objektinformationen in Text- oder XML-Dateien ist eine wichtige Grundlage bei der Entwicklung eines Spiels.

5. Tag
Obwohl ich nun mit Blender schöne Resultate erzielen konnte, möchte ich vorerst weiter suchen. Ich müsste eine grosse Scriptsammlung finden, wo möglichst alle Elemente und Funktionen eines Spiels schon vordefiniert sind. Blender eignet sich meiner jetztigen Einschätzung nach sehr gut zum Erstellen von Figuren und Objekten. Das Programm ist mit einer Art "Game Engine" versehen – diese verfügt aber nicht über Scripts, die speziell für die Entwicklung eines unter Umständen komplexen Spiels vorgesehen sind.
Ich tippe in die Suchmaschine die Worte "game engine open source" ein. Zunächst lande ich bei der Unreal Engine [10]. Die ist zwar gratis verfügbar, im Lizenztext lese ich jedoch, dass die Entwicklung von Spielen für den kommerziellen Gebrauch nicht zugelassen ist. Weiter finde ich Pygame [11]; eine Erweiterung für Python. Ich Installiere die Scripts, beschäftige mich ein wenig damit, finde jedoch keinen richtigen Zugang dazu. Also suche ich weiter. Bei der Game Engine Delta 3D [12] werde ich fündig: Ein umfangreiches Software-Paket und vielversprechende Hinweise, dass unter Anderen der amerikanische Staat zu den Nutzern gehört. Bei der ersten Durchsicht des Handbuchs sehe ich, dass ein sogenannter Game Manager das "Herzstück" darstellt, indem er die Meldungen der Actors (Darsteller) und Components (Komponenten) übermittelt und koordiniert. Ich denke nicht unbedingt daran, ein Kriegsspiel entwickeln zu wollen; von der Komplexität her wird diese Game Engine jedoch auf jeden Fall viele Möglichkeiten bieten.

6. Tag
Ich beginne, Delta zu installieren. Zunächst verwende ich den Windows-Installer und öffne im VisualStudio (obwohl es die Express-Edition ist, werde ich es der Kürze wegen in der Folge so nennen) ein erstes Projekt HelloWorld. Bald merke ich, dass nichts recht geht und der Kompiler ständig Fehlermeldungen ausgibt. Ich überprüfe nochmals die vorher minutiös eingetragenen Umgebungsvariabeln meines Rechners. Dann gehe ich erneut auf das Forum von Delta. Hier finde ich den Hinweis, dass ich vorher den "Platform Software Development Kit" (P SDK) von Microsoft installieren muss. Auch diese Software wird gratis angeboten. Ich muss mich anmelden und Mails mit Downloadlinks öffnen - aber insgesamt geht das alles gut. In der Folge versuche ich stundenlang dieses HelloWorld zu kompilieren. Es gibt nun mit dem SDK weniger Fehlermeldungen, aber immer noch viele. Irgendwann finde ich heraus, dass in den Umgebungsvariabeln "Program Files" steht und nicht "Programme", wie es auf einem deutschsprachigen System sein sollte. Und es wird nicht das letzte Mal sein, dass ich über solche Sachen stolpere.

7. Tag
Zur Abwechslung probiere ich nun die Irrlicht-Engine aus. Mit den entsprechenden Erfahrungen ausgerüstet, trage ich alle Umgebungs- und Kompilervariabeln prompt richtig ein. Innert zehn Minuten steht eine adrette junge Dame in schwarzem Dress als

3D-Figur vor mir. Ich finde im Datenordner ein Bild, in dem die Hinterseite, die Vorderseite und das Gesicht der Figur dargestellt ist. Auch andere Figuren sind in der gleichen Art mit Texturen ausgestattet. Ich frage mich, mit welchem Programm das md2-File, welches die Figur ist, hergestellt wurde (Quake [13]). Ich öffne die Datei mit dem Texteditor und sehe nur kryptische Zeichen.
Das Kompilieren der weiteren 15 Beispiele von Irrlicht macht Spass, da alles tadellos funktioniert. Besonders interessant ist, dass jeweils gewählt werden kann, ob das Beispiel in OpenGL oder in DirectX gerendert wird. Auf meinem relativ schlechten Laptop-Bildschirm sehen die Grafiken in DirectX eindeutig besser aus. Jedoch ist mir als Linux-Benutzer auch bewusst, dass es dort nur OpenGL gibt.

8. Tag
Vom Netz lade ich einige interessante Tools für Irrlicht herunter, mit denen recht einfach ganze Szenen zusammengestellt werden können. Ich probiere einige Sachen aus, ohne das Handbuch genauer zu lesen. Sehr interessant ist ein Objekt, welches Partikelsystem genannt wird. Damit kann beispielsweise fliegendes Laub dargestellt werden. Wie bei einem Mesh-Objekt kann ich es verformen, skalieren und bewegen. Zusätzlich wird die Windrichtung und die Stärke des Windes angegeben. Ebenfalls sehr schön ist die einfache Integration von Sound-Daten. Allgemein scheint sich Irrlicht eher für die Entwicklung eines Adventure- oder eines Shooter-Games zu eignen. Die Engine ansich ist in erster Linie eine Rendering-Engine. Da ich doch eher im Sinn habe, längerfristig ein Strategie-Spiel zu entwickeln, lasse ich die Versuche mit Irrlicht mal so stehen.

9. Tag
Nun traue ich mich doch nochmals ans HelloWorld von Delta heran. Vorher habe ich vom Netz ein paar allgemeine Lehrbücher zu C++ heruntergeladen. Davon gibt es relativ viele, die meisten stammen von deutschen Uni-Professoren; teilweise sind sie etwas alt und behandeln in erster Linie C (die Vorgänger-Sprache von C++). Ich schaue in die C++ Files rein und vergleiche sie mit den Fehlermeldungen aus dem Kompiler. Aus dem einen 500-seitigen Wälzer suche ich relativ zufällig mit der Suchfunktion nach entsprechenden Syntax-Beispielen. Ein paar grundsätzliche Dinge zu C++ werden mir langsam klar und ich kann den Hinweisen aus dem Kompiler immer präziser nachgehen. Ich finde heraus, dass gewisse Teile gar nicht vorhanden sind. Auf dem Delta Forum finde ich verschiedene Hinweise, dass ich für Windows einen Teil der Extensions in einer anderen Version herunterladen muss. Ich finde zudem einen relativ neuen Entwicklungsrelease für diese Extensions. Jetzt stelle ich aus den drei Versionen eine Neue zusammen, in der hoffentlich alle und die richtigen C++ Files enthalten sind. Diese Konfigurationsprobleme und noch Weitere haben mich insgesamt mehrere Tage beschäftigt. Irgendwann lese ich im Forum einen Beitrag der sagt, dass Delta offiziell noch nicht für das VisualStudio 2005 unterstützt wird. Andererseits gibt es

auch Berichte von Leuten, die das Ganze zum Laufen gebracht haben.

10. Tag

Einigermassen zuversichtlich gehe ich nun diesen Berichten nach. Zunächst müssen zwei weitere Programme installiert werden: Der MinGW-Kompiler [14] und der QT GUI toolkit [15]. In einer Anleitung wird gesagt, dass der QT toolkit zuerst gepatcht werden muss, damit er zusammen mit VisualStudio funktioniert. Ich kopiere die Patch-Datei in den Ordner und löse den Vorgang mit einem Doppelklick aus. Offenbar wurden nun einige Dateien neu erstellt und andere überschrieben. Irgendwie scheint das aber mit dem VisualStudio nicht zu funktionieren. Nach längerem Herumprobieren finde ich auf dem Netz in einem Forum einen recht neuen Eintrag, welcher besagt, dass ich die aktuelle Version des QT toolkit's gar nicht mehr patchen muss. Auch wird in diesem Fall gar nicht mit dem VisualStudio kompiliert. Es wird mir dabei wieder einmal bewusst, dass es manchmal viel bringt, zunächst genug Informationen über ein Problem zu beschaffen. Da die Entwicklung von Open Source Software teilweise sehr schnell geht, ist es zudem wichtig, zuerst zu schauen, ob eine Information überhaupt noch aktuell ist. Ich gehe nun den Schritten in dieser aktuellen Anleitung nach. Endlich klappt mal wieder alles und der Rechner kompiliert rund vierzig Minuten vor sich hin. Ich schaue den Meldungen in der Dos-Konsole zu und freue mich jedesmal insgeheim, wenn wieder ein Schritt erfolgreich abgeschlossen wurde. Danach habe ich die brandneueste Software auf meinem Rechner, die ich je hatte. Irgendwie ist das schon ein gutes Gefühl, mit einem Programm zu arbeiten, das eben vor zwei Minuten erstellt wurde. Mit dem QT toolkit können Fenster, Buttons etc. für Windows-Programme erstellt werden. Soweit ich es sehe, funktioniert es einwandfrei. Jetzt habe ich alle Bauteile zusammen, um endlich den Haupt-Editor für Delta kompilieren zu können. Dieser Vorgang dauert nochmals eine halbe Stunde und ich habe erneut den "Hurra-Effekt". Um der Sache noch das Tüpfchen auf das i zu geben, kompiliere ich gleich das HelloWorld-Beispiel. Die Buchstaben "Hello World" erscheinen in einem Fenster, dreidimensional und in grellen Farben. Ich drehe sie mit der Maus etwas herum. Die Bemühungen haben sich soweit gelohnt.

11. Tag

Da der QT toolkit nur unter einer eingeschränkten Lizenz zur Verfügung gestellt wird, teste ich heute eine andere Software, die in etwa dasselbe kann. Meine eigene Vorgabe bei dieser

"Machbarkeits-Studie" ist es ja, nur Programme zu verwenden, welche uneingeschränkt benutzt werden können. Das ist relativ wichtig, damit ich vollständig über die Rechte an den Produkten verfüge, die ich damit herstelle. Crazy Eddie's Graphical User Interface [16] ist mit einer Reihe von Beispielen versehen, die ich alle sogleich kompiliere. Der Stil der Fenster, Buttons und Listen ist etwas von Science Fiction beeinflusst, was aber sicher gut zu einem Spiel passen wird. Irgendwann muss ich noch herausfinden, wie ich eigene Grafiken einbinden kann.

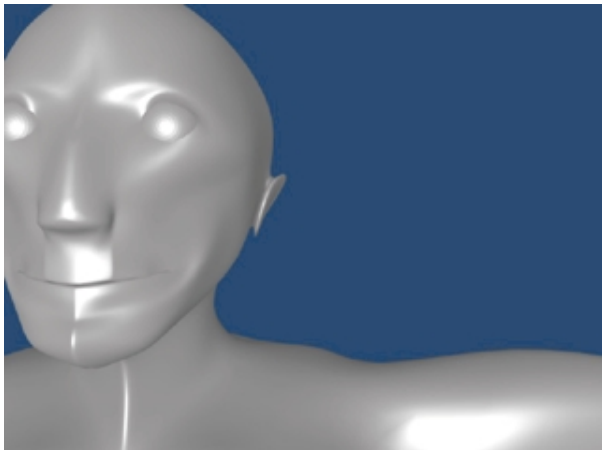
12. Tag

In den folgenden Tagen kompiliere ich alle Tools und Beispiele der Delta-Engine. Ich fasse das hier etwas zusammen. Da die Komplexität dieses Software-Pakets mit einigen hundert Einzelteilen doch recht gross ist, kommt es immer wieder zu kleineren Problemen. Ich beschäftige mich intensiv mit den Kompiler-Voreinstellungen, sehr oft liegt es nämlich nur noch daran, diese im Bezug auf meinen Rechner richtig einzustellen. Gleichzeitig erfahre ich schon einiges über die Fähigkeiten des Programms. So ist beispielsweise die Programmiersprache Python mit integriert und es gibt die Möglichkeit, Netzwerkspiele zu programmieren. Die Figuren in den Beispielen sind durchwegs Soldaten und Militärvehikel – ich habe zwar meine Vermutungen, aus welchem Spiel sie stammen, gehe der Sache aber nicht weiter nach. Sie zeigen exemplarisch verschiedene einfachere Aktionen, meistens mit zwei Figuren pro Beispiel. Es rattert, wenn der Helikopter heranfliegt, knallt, wenn der Soldat schiesst und wenn der Jeep durch die Wüste fährt, wird Staub aufgewirbelt. In mir kommt der Wunsch auf, diese Figuren durch eigene zu ersetzen. Da der C++ Code recht gute Kommentare enthält, sehe ich auch schon, wo die einzelnen Teile der Figuren eingebunden sind.

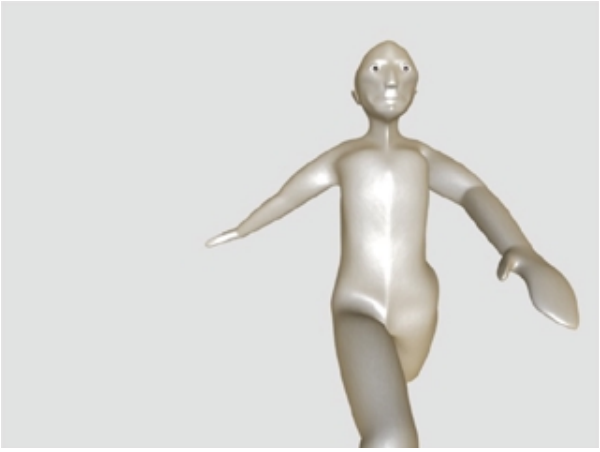
13. Tag

Ich kehre zu Blender zurück und beginne, ein Tutorial durchzuarbeiten, in dem die Erstellung einer Figur von Anfang an bis hin zu einer fertigen Animation erklärt wird (da unter der Blender-Seite eine Vielzahl von Tutorials vorhanden sind, gebe ich hier keinen Link an). Es beginnt mit einer einzelnen Fläche. Da ich mich bislang noch nicht sehr intensiv mit dem Programm auseinandergesetzt habe, wird mich dieses Tutorial rund zehn Tage beschäftigen. Ich versuche mich nicht von der Ästhetik der Beispielfigur beeinflussen zu lassen. Da ich gleichzeitig eine Einladung bekommen habe für eine "Kunst-Präsentation auf dem Internet" [17], möchte ich nun langsam konkrete Fortschritte erzielen. Durch die vorangegangene intensive Beschäftigung mit dem Kompiler bin ich einigermassen zuversichtlich, auch in technischer Hinsicht ein konkretes Resultat erzeugen zu können.
Mir schwebt ein Spiel-Szenario vor, in dem ein prähistorischer Jäger einen Ziegenbock erlegen kann. Daher versuche ich, die Figur in diese Richtung zu gestalten. Er bekommt hochgezogene, überdeutliche Wangenknochen und eine grosse Nase. Der Schädel ist eher flach, dafür sehr lang nach hinten gezogen. Ich versuche

mich ansich in einem relativ "naturalistischen" Stil – gleichzeitig habe ich aber immer noch kleinere Mühen mit der Bedienung des Programms und das Resultat wird dadurch eher in Richtung "Comics-Stil" geführt. Mit den Tagen gewöhne ich mich langsam an das Arbeiten im dreidimensionalen Raum. Grundsätzlich ist es wichtig, oft die Ansichtsweise zu ändern und die Punkte jeweils nur in eine Raumrichtung zu bewegen. Immer wieder kommt es zu unbeabsichtigten Überschneidungen von Flächen.



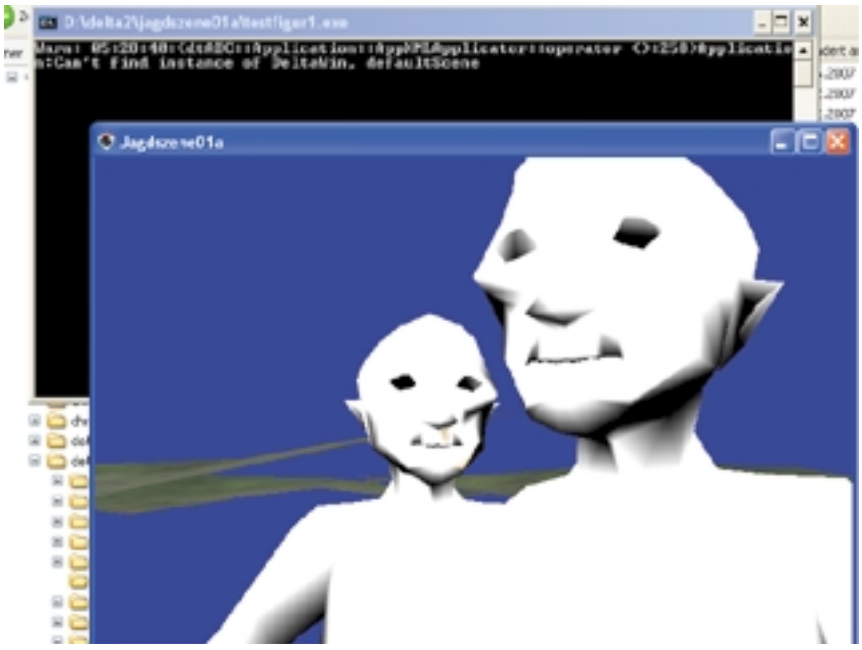
14. Tag
Wir sehen eine erste Ansicht der Figur. Da ich nur die eine Hälfte des Körpers konstruiere und die andere automatisch gespiegelt wird, ist beim Hals und dem Kinn noch eine Naht sichtbar. Recht gut sind mir hingegen die Ohren gelungen, wenn auch diese noch stark abstrahiert sind. Auf dem Internet finde ich Anleitungen, die sich sogar nur mit dem dreidimensionalen Konstruieren des menschlichen Ohrs beschäftigen. Nachdem ich dem Körper und den Augen Texturen bzw. Farben hinzugefügt habe, geht es darum, die Figur zu animieren.
Die Animation einer Figur wird im Fachbegriff auch "rigging" genannt. In einem speziellen Bearbeitungsmodus wird im Innern des Körpers ein eigentliches Skelett erstellt. Danach werden den einzelnen Knochen Teile des umgebenden Meshes zugeordnet. Dem Oberarm beispielsweise wird auch ein Teil der Brust und der Schultern zugeordnet, so dass die Bewegungen später möglichst realistisch wirken. Es werden zusätzlich ein paar spezielle Knochen angebracht; einer ist vor den Augen plaziert und diese werden auf ihn gerichtet. Auf diese Weise können die Augen bewegt werden.
Nachdem das Skelett erstellt ist, kann ich einzelne Bewegungs-Sequenzen aufnehmen und programmieren. Die Figur winkt mit der Hand, geht umher und steht schnaufend da. Innerhalb von Blender kann ich auch Sequenzen kombiniert abspielen: Die Figur geht und winkt gleichzeitig mit der Hand.



15. Tag
Beim Rumstöbern in der Delta-Engine bemerke ich, dass der "Skeletal Mesh Editor" nicht funktioniert. Ich kann ihn zwar kompilieren, aber es hält sich hartnäckig ein Fehler, dessen Ursache ich nicht finde. Also schaue ich die Figuren aus den Beispielen genauer an. Ich finde heraus, dass sie auf der Software "cal3D" [18] basieren. Also lade ich cal3D herunter und versuche, sie zu kompilieren. Leider gelingt das nur teilweise, glücklicherweise gibt es aber schon fertige Programme. Es ist eine Testfigur vorhanden und ich lasse sie laufen. Auch hier zeigt sich eine junge Dame, Cally genannt, diesmal im Steinzeit-Look, was mir besonders gefällt. Mit einigen Mausklicks kann ich sie die vorprogrammierten Bewegungs-Sequenzen ausführen lassen. Sie macht mir einige Karateübungen vor. Sogleich exportiere ich meine Figur aus dem Blender in das Format cal3D. Erstaunlicherweise klappt das problemlos und die Figur läuft vor sich hin. Da sich der Oberkörper dabei nach vorne und hinten bewegt, die Augen aber stehen bleiben, sieht das recht komisch aus. Ich lasse daher in den folgenden Versuchen die Augen weg. Mit den Armen und Beinen gibt es auch Probleme: An den Stellen, wo sich die Haut berührt, entsteht eine Störung in der Darstellung - es ist ein Loch im Mesh sichtbar. Ich baue die Figur im Blender etwas um. Die Arme kommen nun weiter raus, damit sie in einigem Abstand zum Oberkörper sind und ihn beim Gehen nicht mehr berühren. Mit den Beinen bin ich weniger erfolgreich, die Löcher sind bis jetzt geblieben. Zudem bewege ich den Oberkörper beim Gehen nicht mehr, damit ich zu einem späteren Zeitpunkt dann doch Versuche mit den Augen machen kann. Die Figur wirkt nun im Gehen etwas statisch. In Kombination mit den fehlenden Augen hat sie morbide Charakterzüge angenommen. Der prähistorische Jäger gleicht nun Ötzi, wenn er noch am Leben wäre, oder Winnetou in hohem Alter.

16. Tag
Ich wähle aus den Beispielen von Delta eine einfache Szene aus, in der eine Figur mit den Pfeiltasten gesteuert werden kann und eine zweite Figur in einem bestimmten Abstand folgt. Im C++ File suche ich nach der Stelle, wo die Figuren eingebunden werden. Das dort genannte File hat die Endung .rbody und ich merke, dass dieses Format nicht zu cal3D gehört. Ich kann es schliesslich dem

Projekt ReplicantBody [19] zuordnen. Es ist von einem schwedischen Uni-Professor geschrieben. Immer wieder erwähnt er ein Interface, welches ich beim besten Willen aber nicht finden kann. Mit der Zeit merke ich, dass er mit Interface gar nicht eine grafische Benutzeroberfläche meint, sondern ganz einfach dieses rbody-Konfigurationsfile. In diesem File werde die Daten im cal3D-Format integriert und eine kleine Steuerung für die Füsse eingebaut. Es folgt wiederum ein tagelanges Rumprobieren; es dauert ewig, bis die Figur überhaupt in der Szene erscheint, dann nochmals eine Ewigkeit, bis ich die Knochen richtig benannt habe und sich die Figur schliesslich doch bewegt. Mit der Zeit merke ich, dass ich die Anwendung nicht jedesmal neu kompilieren muss, weil die Konfigurationsfiles der Figur dynamisch importiert werden. Ich muss weiter in Blender die ganze Animation deutlich vereinfachen und auch den Mesh mehrfach anpassen.
Ein weiterer wichtiger Darsteller tritt in Erscheinung: Das Gelände. Meine Figur geht nämlich nicht auf ihm, sondern bleibt immer auf einer virtuellen Ebene unter ihm. Es geht um eine offenbar recht komplexe physikalische Rechnung, wo die Füsse mit dem 3D-Gitter des Gländes koordiniert werden sollen. Im Forum von Delta gibt es viele Hinweise darauf, dass andere Leute ähnliche Probleme haben – immerhin. Es macht den Anschein, dass es vorallem dann Probleme gibt, wenn der Mesh der Figur zu komplex ist. So versinkt wohl nicht zufälligerweise auch die britische Kavallerie im Sumpf. Ich lasse meine Figur bis zum Bauch im Sumpf stecken und wende mich der weiteren Realisierung des Projekts zu.



17. Tag
Für einmal geniesse ich die Betrachtung meiner Figuren; so fehlerhaft sie auch sind – es macht Spass, sie im Raum herumgehen zu sehen. Durch die fehlenden Texturen sind die Körper in schwarzweiss gehalten und die Struktur des Meshes ist durch den Export von Blender nach cal3D sehr grob geworden. Nach der Installation

auf einem zweiten Rechner zeigt sich, dass ich eine Anzahl von dll's (sog. Dynamic Linking Libraries) mit in den Ord-ner kopieren muss. Weiter sehe ich, dass der Pfad zum Gelände nicht mehr gefunden wird. Aus diesem Grund wird in der Dos-Konsole, aus der heraus das Programm gestartet wird, permanent eine Warn-Meldung angezeigt. Das nicht mehr wahrnehmbare Gelände wird zu einer Funktion.

Linkliste

GNU Lesser General Public License as published by the Free Software Foundation
1. http://www.fsfeurope.org/index.de.html
2. http://de.wikipedia.org/wiki/C++
3. http://de.wikipedia.org/wiki/Python_%28Programmiersprache%29
4. http://www.google.ch
5. http://www.blender.org
6. http://www.intel.com/cd/ids/developer/asmo-na/eng/254761.htm
7. http://irrlicht.sourceforge.net
8. http://sourceforge.net/index.php
9. http://www.microsoft.com/germany/msdn/vstudio/products/express/visualc/default.mspx
10. http://www.unrealtechnology.com/html/technology/ue30.shtml
11. http://www.pygame.org/news.html
12. http://www.delta3d.org
13. http://www.idsoftware.com/games/quake/quake2/
14. http://www.mingw.org
15. http://trolltech.com/downloads/opensource#qt-open-source-edition
16. http://www.cegui.org.uk/wiki/index.php/Main_Page
17. http://www.netzriss.org
18. https://gna.org/projects/cal3d
19. http://www.vrlab.umu.se/research/replicantbody

Installation
Die Applikation ist zurzeit nur in einer Windows-Version verfügbar. Ich habe sie auf Win XP, Win 2000 und Vista getestet. Entpacken Sie jagdszene01a.zip an einen beliebigen Ort auf ihrem Computer. Beim Entpacken muss die Option "Ordnernamen beibehalten" aktiviert sein. Innerhalb des Ordners "jagdszene01a" sollte sich nun ein Ordner "figur1" befinden. Ist dies nicht der Fall, können Sie ihn auch von Hand erstellen und die Datei cal3d.cfg sowie alle Dateien, die mit "figur8" beginnen darin ablegen.

Start der Applikation: Doppelklick auf testfigur1.exe

Bedienung
Aufwärtspfeil: Figur geht vorwärts
Pfeil links: Figur dreht sich nach links
Pfeil rechts: Figur dreht sich nach rechts
Szene drehen mit gedrückter linker Maustaste
Szene verschieben mit gedrückter rechter Maustaste
Ein- und Auszoomen mit gedrückter mittlerer Maustaste

jagdszene01a ist ein Projekt von Michael Härdi, 2007
http://www.imagedesign.ch