

Virtual Reality im CAVE Projekt: CAVEFlyer

Student/Team

Das Team umfasst einen Studenten (dies minimiert den Kommunikations-Overhead zwischen den Teammitgliedern und gewährleistet dass keine Missverständnisse zwischen den Mitgliedern auftreten können).

Student: Stefan Ortner

Matr.-Nr.: 0357023

SKZ: 086

VRC-Personenkennzeichen: vrpk40

Allgemeine Information

Zu finden ist das Projekt auf der Origin im Home-Verzeichnis des vrpk40 Accounts.

`/home/vrpk/vrpk40/VRC/CAVEFlyer`

Starten lässt sich das Programm mit der ausführbaren Datei `CAVEFlyer`. In diesem Verzeichnis befinden sich auch alle Quellcodedateien um das Programm neu zu übersetzen (dies kann schnell und bequem mit dem Shellscript `cl` erledigt werden). Achtung! Nur für CAVE getestet und empfohlen, für andere VR-Systeme wurde nicht getestet.

Beschreibung

Im Spiel geht es darum, durch einen immer schwieriger zu durchfliegenden Tunnel zu fliegen und möglichst lange die Kollision mit den Tunnelwänden zu vermeiden. Je länger man durchhält, desto mehr Punkte erhält man, weiters kann man seine Score dadurch verbessern in dem man Affenköpfe einsammelt, die in regelmäßigen Abständen im Tunnel erscheinen.

Bedienung

Nach dem Starten des Programms sieht man an der Frontwand wie man das Programm bedient (durch drücken des CAVEButtons Eins startet man das Spiel, während des Spiels kann man durch drücken und halten des CAVEButtons Eins Höhe gewinnen). Am Boden wird eine Stehposition vorgeschlagen. Während des Spiels muss man darauf achten nicht mit den Tunnelwänden zu kollidieren, dies kann man durch drücken und loslassen des CAVEButtons Eins kontrollieren. Wenn man mit dem Tunnel kollidiert endet das Spiel und die erreichten Punkte werden angezeigt. Durch das Drücken des CAVEButtons Zwei gelangt man wieder in den Anfangszustand des Programms.

Realisierung

Entwicklungsumgebung: VisualStudio .NET (7.0), nedit

Entwicklungsplattformen: Windows, Irix

Verwendete API's: KNAVELib/CAVELib, OpenGL

Verwendete Programmiersprache: C++

Verwendete Third-Party Produkte:

- MersenneTwister Klasse: Pseudozufallszahlengenerator (um ein vierfaches schneller als `rand()`), <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>

- TextureManager Singleton: Texturlade und –verwalte Klasse von Christopher Smith (von mir modifiziert um .TGA auf Little und Big Endian Systemen zu laden)

Probleme

Natürlich gibt es kein Softwareprojekt wo keine Probleme auftreten, so auch hier nicht, im Folgenden sind kurz die aufgetretenen Probleme dokumentiert:

- KNAVELib Probleme: Das Registrieren der Callback-Funktionen ist in der KNAVELib nur ohne zusätzliche Parameterübergabe implementiert, da jedoch der Sourcecode verfügbar ist kann man dieses Problem umgehen indem man diese Funktionalität eigenhändig hinzufügt. Achtung! Auch werden in der KNAVELib die vier Wände nicht als eigenständige Prozesse mit eigenständigem Speicherbereich realisiert.
- Little/Big Endian: Beim Laden der Texturen wurden an einigen Stellen mehr als ein Byte auf einmal gelesen, dies führt bei Little und Big Endian System zu unterschiedlichen Resultaten. Dieses Problem wurde durch eine einfache Überprüfung des Systems und wenn nötig vertauschen der Bytes gelöst.
- Compiler: C++ Programme auf IRIX grundsätzlich mit dem CC Compiler übersetzen!

Mögliche Verbesserungen/Erweiterungen

Den Tunnel als Sharedmemory-Objekt anlegen und nur noch wenn das `CAVEMasterDisplay()` aktiv ist updaten.

Einen ausgeklügelteren Tunnelverlauf-Algorithmus implementieren (muss das Interface `ICourseAlgorithm` implementieren). Algorithmussystem wurde im Strategy Pattern realisiert (ermöglicht rasches auswechseln der Algorithmen).

Troubleshooting

Ab und an kann es vorkommen dass der Tunnel nicht synchron auf allen vier Wänden des CAVE läuft, dies kann man durch einen Neustart beheben.