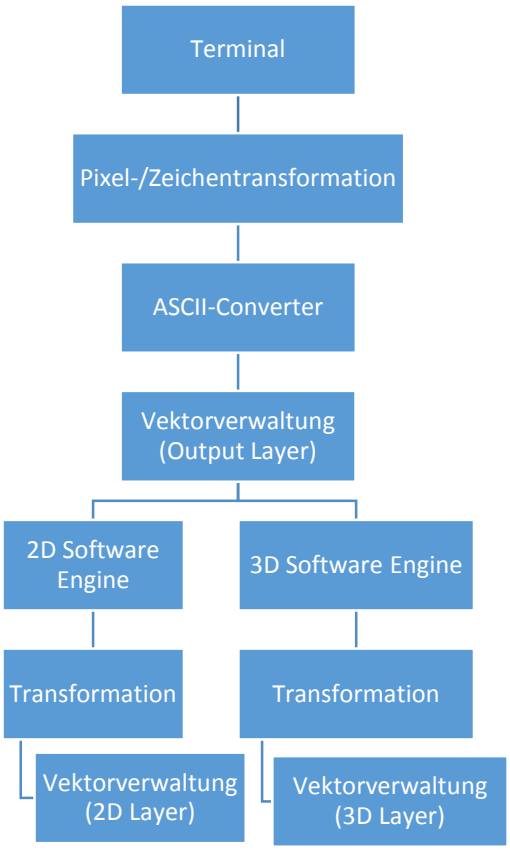
TE3D – Eine 3D Terminal Engine

TE3D soll es ermöglichen, Grafiken im Terminal anzuzeigen (sowohl 2D als auch 3D). Die 3D-Engine wird eigenhändig geschrieben, auf Hardwareunterstützung (z. B. Grafikkarte) wird bei dem Rendering-Prozess verzichtet. Die Engine soll die gängigsten Funktionalitäten unterstützen wie Modellverwaltung und Kamerasysteme. Die 2D-Grafikimplementierung wird auch mithilfe von Vektoren realisiert, wobei nur die x- und y-Komponente dieser verwendet wird.
Nach der fertigen Berechnung der Vektoren werden diese mithilfe eines „ASCII-Art"-Converters in Zeichen umgewandelt, die dann ins Terminal geschrieben werden. Dabei soll auch Farbe unterstützt werden (mithilfe von Terminal-Steuerzeichen).

TE3D soll so wenig externe Bibliotheken verwenden wie möglich und plattformunabhängig sein.
TE3D soll als Bibliothek erstellt werden, damit das Projekt überall eingebunden werden kann.

Design

TE3D ist in verschiedene Layer aufgeteilt, in ihrer Gesamtheit als Pipeline bezeichnet. Die Hauptlayer in der Pipeline sind:



- Terminalausgabe
  Die Terminalausgabe wird mit einer Renderfunktion (`void TE3D_Present()`) bewerkstelligt, die einen Zeichenpuffer anspricht, der ein Array von ASCII-Zeichen (Datentyp `char`) und deren entsprechende Farbe beinhaltet (Struktur `TE3D_Surface`). Nach dem Rendern wird die Position des Terminalstreams zurückgesetzt und es kann erneut gerendert werden. Die Renderfrequenz kann vom Benutzer angegeben werden.

- Pixel-/Zeichentransformation
  Ermöglich das nachbearbeiten der fertig gerenderten Zeichen nach dem Konvertieren (zum Beispiel für Nebeleffekte etc.). Diese Transformation ist optional. Mithilfe des Members `bool TE3D_DoPixelTransform` der `Struktur TE3D_Surface` kann die Ausführung der Transformation gesetzt werden.

- ASCII-Converter
  Der ASCII-Converter ist für die Konvertierung von Vektoren in Text verantwortlich. Die Funktion
  `void TE3D_ASCII_Convert (*TE3D_Vector4g, int count, struct TE3D_Surface target)`
  wird die Hauptfunktion des Converters sein.

- Vektorverwaltung (Output Layer)
  Die Vektorverwaltung im Output Layer ist für das Verwalten der Vektoren zuständig, nachdem sie von der 2D- und 3D-Engine transformiert wurden und enthält zusätzlich einen Puffer zur Speicherung der zu rendernden Linien / Flächen. Diese separate Verwaltung ist wichtig, da die Transformation der Vektoren in den Engines auch getrennt abläuft aufgrund der sonst möglichen Vermischung der Transformationen.
  An dieser Stelle werden die Vektoren aus den Engines zusammengeführt.
  Vektoren im Output Layer und in den Engines sind vierdimensional (`struct TE3D_Vector4f`). Andere Dimensionen von Vektoren (`TE3D_Vector3f`, `TE3D_Vector2f`) sind im Framework enthalten, allerdings werden diese nicht als Eingabeparameter für die Transformationslayer von TE3D verwendet (Ausnahme: 2D-Transformationen, hier werden 3-dimensionale Vektoren und 3x3-Matrizen verwendet), die Rechenregeln für alle definierten Matrizen und Vektoren sind aber implementiert (Zum Beispiel `TE3D_Matrix4x4f_mul(TE3D_Matrix4x4f matrixA, TE3D_Matrix4x4f matrixB)`, welche das Matrixprodukt $M_A * M_B$ bildet.).

- 2D Software Engine
  Zuständig für die Verwaltung von zweidimensionalen Zeichenfunktionen (wie das Malen einer Linie). Die 2D-Software-Engine besitzt (wie die 3D-Engine) eine eigene Pipeline. Um Hintergründe und andere Effekte zu ermöglichen, gibt es zusätzlich das s. g. Pre- und Postrendering (Rendern bevor die 3D-Vektoren gezeichnet werden oder anschließend). Dabei wird die Tiefenkomponente (z-Komponente) der Vektoren auf möglichst nahe der Projektionsebene und beim Postrendering auf möglichst fern gesetzt.

    - Transformation
      Hier werden alle Vektoren des 2D-Raumes transformiert. Dazu wird eine Haupt-3x3-Matrix verwendet (`struct TE3D_Matrix3x3f`). Mehrere Transformationen hintereinander werden durch ausmultiplizieren der entsprechenden Transformationsmatrizen hintereinander erreicht.

    - Vektorverwaltung (2D Layer)
      Regelt das Vektorensystem in der 2D Engine. Im System werden zwei Listen genutzt, die jeweils Vektoren für das Pre- und Postrendering beinhalten.

- 3D Software Engine
  Beinahe identischer Aufbau zur 2D-Engine, doch anders als diese kümmert sich die 3D-Engine um die Transformation dreidimensionaler Vektoren. Sie besitzt dieselben Pipeline-Layer mit leichter Abänderung.

    - Transformation
      Getrennt von der 2D-Transformation arbeitet diese Engine mit 4x4-Matrizen (TE3D_Matrix4x4f). Auch hier wird nur eine einzige Hauptmatrix zur Transformation verwendet.

- Vektorverwaltung (3D Layer)
  Dieses Verwaltungssystem benutzt nur eine einzige Liste zur Verwaltung der Vektoren. Diese Liste enthält nicht die Vektoren selbst, sondern „Modelle“, Gruppierungen einzelner Vektoren. Das macht das Verwalten einzelner 3D-Objekte möglich und dessen eigene Transformation.

Zusätzlich kommen zu dieser TE3D-Pipeline zusätzliche Strukturen hinzu für Kamerasysteme, Transformationsobjekte etc.

Quellcodestil

- Jede Codedatei enthält eine Kommentarkopfzeile mit den Namen der Projektmitglieder und der Lizenz des Projekts (GNU GPL 3 oder höher).
- Verwendung von „`//`“ für wenig-zeilige Kommentare.
- Kommentare auf Englisch.
- Sinnvolle Kommentare auch innerhalb von Funktionen zur besseren Übersicht der anderen Mitglieder.
- Beschreibende Kommentare über den jeweiligen Funktionen (Beschreibung der Funktionalität, Parameter, Rückgabewert etc.). Ist die Funktion öffentlich, wird die volle Beschreibung in die Headerdatei zur Deklaration geschrieben und nur die Funktionalität in die Codedatei. Ist die Funktion nicht öffentlich (bzw. kommt nicht in einer Headerdatei vor), wird die volle Beschreibung über die Definition der Funktion geschrieben.
- Pointerparameter zu Strukturinstanzen von „Pseudo“-Memberfunktionen von Strukturen müssen nicht dokumentiert werden.
- „`{`“ in neuer Zeile.
- „`TE3D`“ als Präfix für alle öffentlichen Funktionen.
- Header aus eigenem Ordner (`#include "abc"`) liegen vor den Einbindungen der Systemheader (`#include <abc>`).

Projektstruktur

● Öffentliche Module ● Kernmodule ● Hilfsmodule

Modul	*Dateien*	*Beschreibung*
Konsolenbibliothek	console.h console.c	Stellt die Ausgabefunktionen für Konsole/Terminal bereit.
Transformationsbibliothek	transformation.h transformation.c	Transformationsfunktionen für Vektoren und Matrizen.
ANSII Converter	ascii_converter.h ascii_converter.c	Konvertiert die Vektoren und Grafiken in ASCII-Art.
Kernbibliothek	core.c	Verbindet alle Funktionen zur Pipeline.
Verkettete Listen	list.h list.c stack.h stack.c	Stellt Listen zur generischen Typverwaltung bereit.
Vektoren Output Layer	graphics.c graphics.h	Verwaltet die Vektoren, bevor diese in den ASCII-Converter gelangen und stellt 2D- und 3D-Funktionen sowie Strukturen bereit.

Einteilung

3D-Engine	Mischa Krüger
2D-Engine und Vektormanagement im Output Layer	Frank Zimdars
ASCII-Art-Converter	Ammar Al-Qaiser
Kern und verkettete Listen	Gordon Kemsies