Dashboard

Bei **OpenGLES** handelt es sich um ein mächtiges Framework, das nicht immer besonders intuitiv zu handhaben ist - die Lernkurve ist entsprechend hoch. Da es sich bei dieser Übung nur um eine Einführung in das Thema handelt, werden wir den Großteil des **OpenGLES**-Codes in Funktionen kapseln, um die Handhabung zu erleichtern.

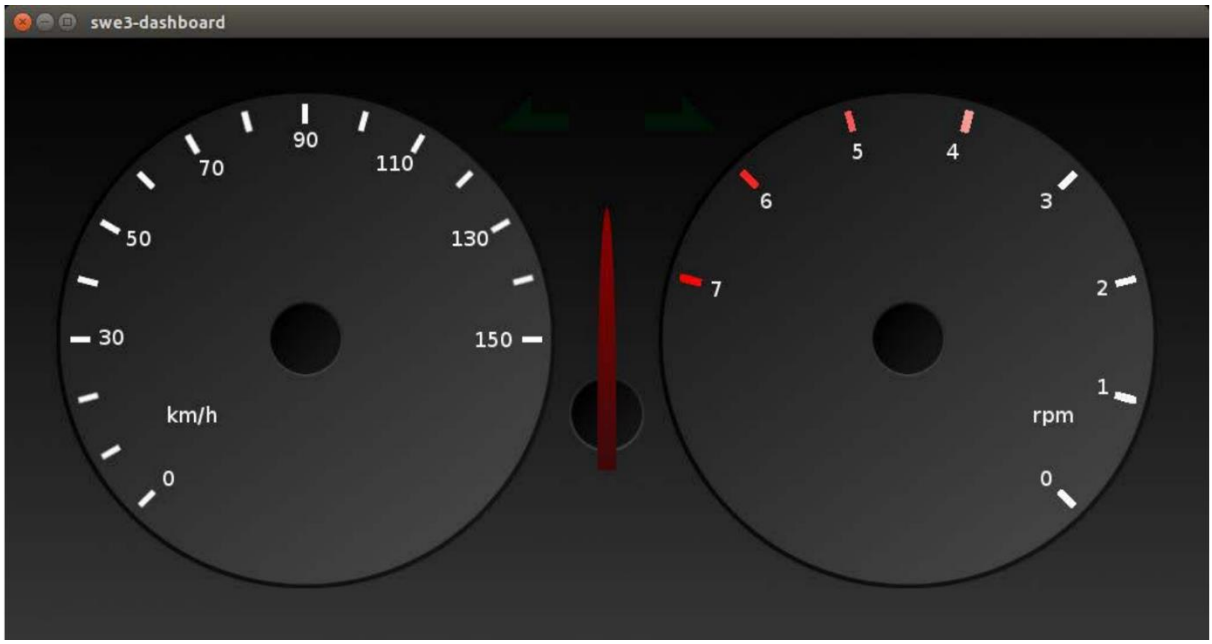
Aufgabe 1: CPU-Last

1. Öffnen Sie ein Terminal-Fenster und führen den Befehl

   ```
   top
   ```

   aus. Jetzt sollten Sie eine etwas kryptische Liste mit aktuell aktiven Prozessen sehen, die nach der CPU-Last, die selbige verursachen, sortiert sind.

2. Importieren Sie das existierende **Projekt Blatt4_Aufgabe1**, übersetzen Sie es und führen Sie es aus. Bei Erfolg ist das abgebildete Fenster mit einer sehr einfachen Kombi-Instrumentengrafik und einer (falsch platzierten) Nadel in der Mitte des Bildes zu sehen.

   Lassen Sie das Fenster geöffnet und werfen erneut einen Blick auf die Anzeige des **top**-Befehls. An erster Stelle sollte jetzt der Prozess **Blatt4_Aufgabe1** zu sehen sein, der den Prozessor enorm beansprucht. Bei einem Laptop sollte sich auch der Lüfter entsprechend schnell bemerkbar machen.

3. Warum beansprucht eine so simple Applikation den Prozessor so stark?

4. Werfen Sie einen Blick auf den vorgegebenen Code, versuchen Sie den Ablauf grob zu verstehen.

5. Wie oft wird das Bild pro Sekunde neu gezeichnet? Wodurch wird die Bildrate limitiert?

   Hinweis: Verständnisfrage, Messwerte spielen hier keine Rolle!

6. Machen Sie sich Gedanken darüber, wie oft ein Bild pro Sekunde gezeichnet werden muss.

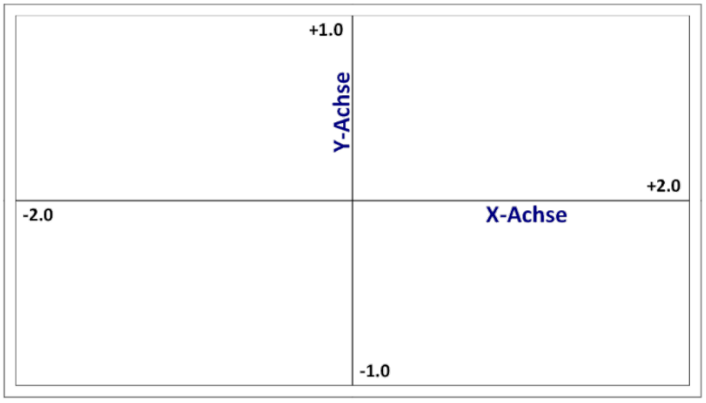
7. Muss das Bild in dem vorgegebenen Code überhaupt neu gezeichnet werden?

8. Finden Sie eine Möglichkeit, die CPU-Last auf verträglichere Werte zu senken.

   Hinweis: Dazu reicht ein einzelner Befehl, den Sie aus der CAN-Parser Übung bereits kennen sollten.

Aufgabe 2: Translation und Rotation

OpenGL-Koordinatensystem



Bearbeiten Sie die in der Liste aufgeführten Punkte:

1. Erzeugen Sie ein neues Projekt mit dem Namen **Blatt4_Aufgabe2** und kopieren Sie die Quelldateien ***.cpp** bzw. ***.h** aus **Blatt4_Aufgabe1** in das neue Projekt.

Hinweis: Für OpenGLES sind einige Libraries erforderlich, die Sie ***in allen*** neuen Projekten hinzufügen müssen, damit der Build-Vorgang fehlerfrei durchgeführt werden kann. Öffnen Sie hierfür

Properties ➔ C/C++ General ➔ Paths and Symbols ➔ Libraries

Nun fügen Sie folgende Libraries hinzu:

pthread
rt
EGL
X11
GLESv1_CM
m
png

2. Beschäftigen Sie sich mit den Befehlen **glTranslate**() und **glRotate**().
3. Werfen Sie erneut einen Blick auf den vorgegebenen Code, wozu dienen die Befehle **glPushMatrix**() und **glPopMatrix**()?
4. Verschieben Sie die Nadel an ihre richtige Position auf der km/h-Anzeige.
5. Rotieren Sie die Nadel um z. B. 45 Grad nach links oder rechts. Wählen Sie andere Werte und probieren es erneut. Was fällt Ihnen auf? Korrigieren Sie das beobachtete Verhalten.

Hinweis: Um welchen Punkt rotiert die Nadel? Überlegen Sie, wie sich die Orientierung des Koordinatensystems der Nadel ändert, nachdem Sie eine Rotation durchgeführt haben.

6. Lassen Sie die Nadel auf 0 km/h und danach auf 150 km/h zeigen.

Hinweis: Die Markierungen sind gleichmäßig verteilt.

7. Implementieren Sie die Funktion **GLfloat kmh2deg(GLfloat kmh)**.Die Funktion soll einen km/h-Wert entgegennehmen und dafür einen Wert in Grad zurückliefern, mit dem die Nadel auf dem Tachoblatt passend gedreht werden kann.

Aufgabe 3: Mehrere Objekte zeichnen

1. Erzeugen Sie ein neues Projekt mit dem Namen **Blatt4_Aufgabe3** und kopieren Sie die Quelldateien *.**cpp** bzw. ***.h** aus **Blatt4_Aufgabe2** in das neue Projekt. Passen Sie die erforderlichen Libraries in den Projekteinstellungen an.

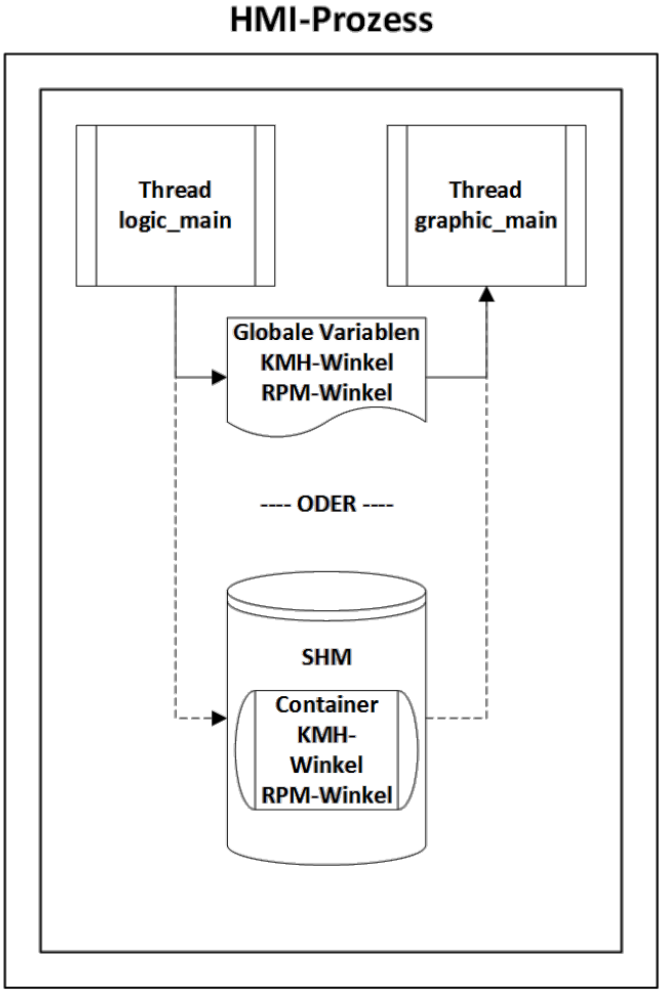
2. Zeichnen Sie eine zweite Nadel in die Mitte des Bildes.

Hinweis: Nur ein einzelner Befehl ist dazu nötig!

3. Verschieben Sie die Nadel in die rpm-Anzeige auf der rechten Seite. Worauf muss dabei geachtet werden? Welche Rolle spielen **glPushMatrix**() und **glPopmatrix**()?
4. Lassen Sie die Nadel auf 0 rpm und danach auf 7 rpm (7000) zeigen.
5. Implementieren Sie die Funktion **GLfloat rpm2deg(GLfloat rpm).**

Aufgabe 4:

Systemübersicht



1. Erzeugen Sie ein neues Projekt mit dem Namen **Blatt4_Aufgabe4** und kopieren Sie die Quelldateien ***.cpp** bzw. ***.h** aus **Blatt4_Aufgabe3** in das neue Projekt, passen Sie zusätzlich die Libraries in den Projekteinstellungen an.

2. Beschäftigen Sie sich mit dem Inhalt der Header **gles**.h und **tile**.h. Wie sind diese aufgebaut? Wozu benötigt man Funktionsprototypen?

3. Erstellen Sie vier leere Dateien namens **hmi**.cpp und **hmi**.h sowie graphic.cpp und **graphic**.h, die sich im gleichen Verzeichnis wie main.cpp befinden.

4. Lagern Sie die Funktionen **kmh2deg**() und **rpm2deg**() aus Ihrer **main**.cpp aus.

5. Erstellen Sie eine Threadfunktion **logic_main**() in Ihrer **hmi**-Bibliothek. Diese erzeugt km/h- und **rpm**-Werte und wandelt diese mit **kmh2deg**() und **rpm2deg**() in Winkel um.

6. Lagern Sie Ihre Grafik-Funktion aus Ihrer **main**(void) in die Threadfunktion **graphic_main**() aus. Diese ist in Ihrer **graphic**-Bibliothek zu erstellen. Die main.cpp sollte danach nur noch aus wenigen Befehlen bestehen.

7. Als Nächstes soll **graphic_main**() und **logic_main**() in jeweils einem eigenen Thread laufen (siehe Abbildung Systemübersicht).

> **Hinweis**: Code aus vorherigen Übungen kann und sollte genutzt werden.

8. Welche Schritte sind nötig um **km/h**- und **rpm**-Winkel aus einem anderen Thread heraus setzen zu können? Passen Sie **graphic_main**(). entsprechend an!

9. Lassen Sie die Funktion **logic_main()** in einem zweiten Thread laufen und simulierte km/h- sowie **rpm**-Werte erzeugen und in Winkel umrechnen.

> **Hinweis**: Zwischen der Logik und der Grafik werden nur Winkelwerte ausgetauscht. Die Umrechnungsfunktionen werden nicht in der **graphic_main**() ausgeführt.

10. Ihre Grafik Funktion mit zugehöriger Grafik Threadfunktion **graphic_main**() lagern Sie entsprechend in Ihre Grafik-Bibliothek

Viel Spaß!