2 Skelettdarstellung

Als Grundlage für die Avataranimation soll ein Skelett dienen. Diese Aufgabe beschäftigt sich damit, eine Skelettbeschreibung aus einer ASF-Datei zu lesen, anzuzeigen und zu animieren.

Nutzen Sie den mitgelieferten Code als Grundgerüst für alle weiteren Aufgaben. Für alle Übungsteile werden Lösungen nach den Abgabeterminen veröffentlicht. Sollten Sie eine Teilaufgabe nicht vollständig umsetzen können, nutzen Sie diese Lösungen für alle weiteren Aufgaben. Grundsätzlich steht es Ihnen frei, die Deklarationen von Klassen und Funktionen zu verändern. Beachten Sie jedoch, dass die Musterlösungen dann ggfs. nicht mehr kompatibel sind.

Der Code basiert auf dem an der Professur entwickelten `cgv`-Framework. Dabei handelt es sich um mehrere Programmbibliotheken, mit denen eine Vielzahl gängiger Algorithmen und Datenstrukturen zur Programmierung von computergrafischen Anwendungen abgedeckt wird. Funktionalität wird im `cgv`-Framework mittels Plugins bereitgestellt, die von einer zentralen Anwendung, dem `cgv-viewer`, aus geladen werden. Er setzt sich aus zwei Teilen zusammen: der 3D-Szene auf der linken Seite und Einstellungsfeldern auf der rechten Seite. Jedes Plugin kann eine beliebige Anzahl solcher Kontrollelemente definieren. Alle in der Anwendung registrierten GUI-Objekte sind in Reitern angeordnet, die sich unten rechts befinden. Für die Navigation in der 3D-Szene liegt dem Viewer das `stereo_view_interactor`-Plugin bei, das eine Maussteuerung erlaubt.

Das vorliegende Projekt besteht aus einem Plugin, das ein Demo-GUI-Objekt registriert, und den `cgv`-Bibliotheken in Binärform. Außerdem liegen Projektdateien für VisualStudio sowie zur Erstellung mittels CMake eine `CMakeLists.txt` bei. Falls Sie unter Linux entwickeln möchten, müssen Sie die `cgv`-Bibliothek, den Viewer und die Plugins selbst kompilieren. Führen Sie dazu das Skript `buildLinux.sh` im `framework`-Ordner aus. Dieses lädt den Quellcode des Frameworks herunter und kompiliert die notwendigen Bibliotheken. Die Entwicklung unter Mac OSX wird momentan nicht unterstützt. Engagierte Studenten sind jedoch dazu angehalten, es mittels des CMake-Systems zu probieren.

Machen Sie sich mit dem Demo-GUI-Objekt vertraut, indem Sie den Quellcode in `CGVDemo.cpp` analysieren. In dieser Datei sind Codebeispiele zur GUI-Programmierung im `cgv`-Framework enthalten, die für weitere Aufgaben nützlich sein können. Das Objekt wird im Konstruktor von `Initializer` in `main.cpp` registriert. Nachdem Sie sich mit der Funktionalität vertraut gemacht haben, können Sie seine Registrierung auskommentieren.

2.1 Vorberechnung von relevanten Matrizen (4 Punkte)

Die Klasse `Skeleton` beinhaltet bereits eine Methode `fromASFFile(const std::string&)`, die Skelettdaten aus einer ASF-Datei lädt. In diesem Dateiformat wird die Struktur eines Skeletts beschrieben. Dabei sind sowohl Attribute enthalten, die für eine statische Visualisierung des Skeletts notwendig sind (z.B. die globale Richtung und Länge von Knochen) als auch animationsrelevante Informationen (z.B. die Freiheitsgrade der Knochen). In `Bone.h` wird beschrieben, welche Attribute aus der ASF-Datei gelesen werden. Weitere Informationen zum ASF Dateiformat finden Sie bspw. unter `http://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/ASF-AMC.html`.

Aktivieren Sie in `main.cpp` den `SkeletonViewer`. Danach taucht in der GUI ein Reiter auf, mit dem die Skelettvisualisierung gesteuert werden kann. Über den entsprechenden Button kann ein ASF-Skelett geladen werden. Eine Visualisierung der Hierarchie erscheint danach in der oberen Baumansicht. Sie können zum Testen die mitgelieferte Datei `jump.asf` nutzen.

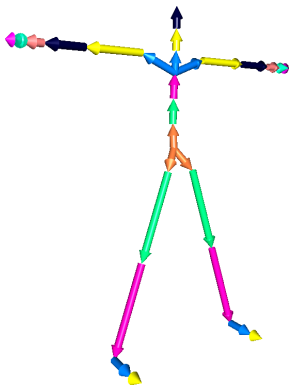
Obwohl eine statische Visualisierung allein anhand der globalen ASF-Attribute möglich ist, erlaubt dieses Vorgehen keine Animation. Deswegen soll keine direkte Visualisierung der globalen Attribute erfolgen (wie z.B. die Knochenrichtung und -länge). Berechnen Sie in einem Vorverarbeitungsschritt in `Bone::calculate_matrices()` die Matrizen `orientationTransformPrevJointToCurrent` und `translationTransformCurrentJointToNext` (2 Punkte). Die Bedeutung dieser Matrizen ist aus den Kommentaren in `Bone.h` ersichtlich. Implementieren Sie außerdem die folgenden Methoden, deren Funktionalität ebenfalls aus der Headerdatei hervorgeht (2 Punkte):

- `Bone::calculate_transform_prev_to_current_without_dofs()`
- `Bone::calculate_transform_prev_to_current_with_dofs()`

Die enthaltenen Freiheitsgrade eines Knochens beinhalten die Funktion `calculate_matrix()`, die Sie für diese Aufgabe verwenden können.

2.2 Statische Skelettvisualisierung (4 Punkte)

In der Klasse `SkeletonViewer` finden Sie ein Grundgerüst für einen Skelett-Betrachter. Implementieren Sie die Funktion `draw_skeleton_subtree()`, die einen Teilbaum einer Skeletthierarchie rekursiv visualisieren soll. Der übergebene Knochen definiert die Wurzel des Teilbaums. Die übergebene Matrix definiert eine Modelltransformation aus dem globalen Koordinatensystem zum lokalen Koordinatensystem des Elternknochens. Für eine statische Visualisierung müssen die Freiheitsgrade noch nicht beachtet werden. Beginnen Sie mit einfachen Liniensegmenten (2 Punkte). Diese Visualisierungform soll anschließend so erweitert werden, dass die Hierarchie deutlich wird. Ersetzen Sie dabei die Linien durch Zylinder mit Pfeilspitzen an ihren Enden. Die so entstandenen Glyphen sollen zum Kindelement zeigen und je nach Hierarchiestufe eine andere Farbe besitzen (2 Punkte). Sie können dafür die Funktion `cgv::render::context::tesselate_arrow()` verwenden. Das benötigte `context`-Objekt erhalten Sie aus der `draw()`-Methode. Das Resultat könnte etwa so aussehen:



2.3 Dynamische Skelettvisualisierung (3 Punkte)

Stellen Sie die statische Visualisierung auf eine dynamische Visualisierung um. D.h., beachten Sie die enthaltenen Freiheitsgrade. Diese können Sie über die GUI für jeden Knochen einstellen, sobald dieser angeklickt wurde. Beachten Sie, dass einige Knochen keine Freiheitsgrade haben (1 Punkt).

Stellen Sie außerdem die in den Knochen definierten Maximal- und Minimal-Winkel dar, indem Sie an den Gelenken entlang der lokalen Rotationsachsen jeweils ein Kreisscheibensegment rendern, das vom Minimal- bis zum Maximalwinkel reicht, sowie eine Anzeige für den aktuellen Winkel (2 Punkte). Dabei können Sie die Klassendeklarationen und -definitionen beliebig erweitern.

2.4 Skelettanimation (3 Punkte)

Die Klasse `Animation` beinhaltet Funktionen, mit denen AMC-Animationsdateien geladen werden können. Erweitern Sie die Methode `SkeletonViewer::load_animation()` um das Laden der Datei und `SkeletonViewer::timer_event()` um das Abspielen der Animation. Die mitgelieferte Animation hat eine Framerate von 120 fps. Sie können diese Framerate als konstant für alle Animationen annehmen. Erlauben Sie außerdem das Starten und Stoppen der Animation mit den Funktionen

`SkeletonViewer::start_animation()` und `SkeletonViewer::stop_animation()`. Erweitern Sie gegebenenfalls die Deklaration der Animationsklasse.

Testen Sie Ihre Implementierung mit der mitgelieferten `jump.amc` oder einer anderen Animation aus der CMU MoCap-Datenbank (`http://mocap.cs.cmu.edu`). Beachten Sie, dass einige Animationen spezifische Skelette benötigen.

Abgabe

Verpacken Sie alle Quelldateien des `src`-Ordners in ein ZIP-Archiv und laden Sie sie in Opal hoch. Achten Sie darauf, dass der Quellcode auf den zur Abnahme verwendeten Rechnern lauffähig ist (Windows, Visual Studio $\geq$ 2015). Zum Testen stehen Ihnen die Rechnerpools der Fakultät zur Verfügung. Sprechen Sie ggfs. mit Ihrem Tutor.