Zusammenfassung / Abstract

<Kurze Darstellung der wesentlichen Aspekte der Arbeit. Dem potentiellen Leser soll geholfen werden, schnell zu entscheiden, ob die Arbeit für ihn interessant seine könnte. Da das Abstract auch die Ergebnisse der Arbeit darstellt, kann es erst zum Schluss geschrieben werden.>

Keywords

OpenCV, OpenGL, Augmented Reality

Arbeitsplan

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Einführung

Bei Augmented Reality geht es um die Erweiterung der Realität, beispielsweise durch eingeblendete Informationen, Bildern, 3D-Modellen usw.

Ziel dieser Arbeit ist es, ein auf den Bibliotheken OpenGL und OpenCV Framework für Augmented Reality zu erstellen. Das Framework soll einfach einzurichten und zu bedienen sein, sodass es ohne OpenGL Kenntnisse und ohne compilen und installieren von Bibliotheken direkt eingesetzt werden kann.

Aufgabenanalyse

<Genauere Formulierung der Aufgabenstellung, Analyse der Randbedingungen, Definition eigener Abgrenzungen, ....>

Die Aufgabe dieser Projektarbeit ist die Analyse des Tools *Menpo*, ob es dazu geeignet ist aus synthetischen Silhouetten ein 2D-Personenmodell zu erstellen. Dabei ist die Hauptabgrenzung schon dadurch gegeben, dass *Menpo* genutzt werden soll.

Dabei soll auch untersucht werden, ob *Menpo* mit Snakes umgehen kann, bzw. die im *Active Contours Paper [1]* eingeführten Methoden beherrscht.

Aufgabe der Projetarbeit ist das Implementieren eines Frameworkes zum Einsatz für Augmented Reality Anwendungen. Für Augmented Reality wird der Zugriff auf eine Kamera sowie Algorithmen zur Projektion und Rückrechnung der Welt in die Computer Szene und andersherum benötigt. Für diese beiden Zwecke soll die freie Computer-Vision Bibliothek OpenCV genutzt werden. Zusätzlich müssen noch 2D- und 3D Grafiken angezeigt werden können, wofür OpenGL genutzt werden soll.

Ziel der Ursprünglichen Aufgabe war es, als Render-Backend Vulkan anstatt OpenGL zu verwenden. Da das Framework im Labor im Raum E-U002 eingesetzt werden soll, und hier nur ältere Rechner ohne Vulkan Unterstützung zur Verfügung stehen, wird OpenGL verwendet.

Lösungsansatz

<Vergleich möglicher Lösungsansätze, Begründung des gewählten Lösungsansatzes, weitere Implikationen des gewählten Ansatzes, ...>

Verwendung von Menpo und die Analyse welche Funktionen und Fähigkeiten Menpo besitzt und ob es dazu fähig ist die oben genannte Aufgabenstellung zu erfüllen.

Dabei wurde festgestellt, dass es in Menpo viele verschiedene Pakete zur Lösung verschiedenster Probleme (Landmark Localization, Affine Image Alignment...) gibt.

Implementierung

<Wie wurde die Aufgabe konkret gelöst, Beschreibung der Implementierungsdetails. Quelltextzitate nur falls sie zum Verständnis der Lösung beitragen.>

Der Folgende Abschnitt soll eine Übersicht aller Klassen des Frameworks und deren Funktion geben.

## Klassenübersicht

### Bibliotheken

Als Mathe-Bibliothek kommt GLM zum Einsatz. Dieses ist sehr nah an den OpenGL GLSL Shadern und bringt alle für das Framework nötigen Funktionen und Klassen mit sich. Am meisten werden hier glm::vec3 für Vektoren, glm::quat für Quaternionen (Drehungen) und glm::mat3 bzw. glm::mat4 für Matrizen verwendet.

### Window

Die Klasse Window bildet die Hauptklasse des Frameworks. Sie erstellt ein neues Fenster und verwaltet den MainLoop der Anwendung. In diesem MainLoop werden alle vorhandenen Objekte auf dem Bildschirm gerendert. Um den MainLoop zu starten, muss die Funktion start() aufgerufen werden. Diese Funktion wird erst beendet, wenn das Fenster geschlossen wird.

Um auf Fluss Anwendung bestimmen zu können besitzt Window den Callback onUpdate, welche vor jedem Render-Vorgang aufgerufen wird. Hier können beispielsweise neue Objekte hinzugefügt, verändert und gelöscht werden. Im Window für jeden Renderdurchgang die Vergangen Zeit zwischen dem Letzten und dem aktuellen Frame berechnet und als deltaTime übergeben. Dies kann dazu verwendet werden, Prozessor- und Lastunabhängige Animationen (z.B. Rotation oder Verschiebung eines Würfels) zu implementieren.

Der Hintergrund des Fensters muss manuell jeden Frame geändert werden, sonst bleibt dieser gleich. Hierzu gibt es die Methode setBackground(), welche eine Textur (im OpenCV Format cv::Mat) erwartet.

Um Tasteneingaben verarbeiten zu können wird die letzte gedrückte Taste als integer Wert vom Window gespeichert und kann via getLastKey() abgefragt werden.

Um Punkte vom Screen in den Worldspace und anderst herum umzurechnen, bietet Window die Funktionen worldToScreenPoint() und screenToWorldPoint(). Um Den Abstand zweier Punkte (Beispielsweise den Radius eine Kreises) zwischen den Koordinaten-Systeme umzurechnen, bietet Window die Funktionen screenToWorldDistance() und worldToScreenDistance().

### Scene

Jedes Fenster enthält genau eine Scene, welche alle Objekte in einer Liste (vector objects) enthält.  
Um auf ein Objekt der Szene zuzugreifen und es zu verändern, kann folgender Code verwendet werden:

lwar::Object3d& monkey = scene.objects.at(0);

### Camera

Die Klasse Camera verwaltet je eine Webcam. Es ist eine Kapselung der Klasse cv::VideoCapture mit vereinfachten Methoden zum Einstellen der Auflösung. Zum Verwenden einer Kamera muss diese nach dem Erstellen mit der Funktion init() initialisiert werden. Via des bool isOpened kann abgefragt werden, ob die Kamera erfolgreich geöffnet werden konnte. Die Kamera wird nicht automatisch erstellt und muss zusätzlich zum Window immer beim Start des Programmes vor der start() Funktion erstellt werden.

### Light

Jede Szene hat genau ein Licht in Form des structs Light. Hierbei kann die Farbe, Position und Stärke des Lichts der Szene definiert werden. Das Licht ist standardmäßig definiert, an der Stelle (1,1,1) platziert und muss nicht zwangsweise verändert werden.

### Object3d

Object3d stellt die Basisklasse für alle Objekte in der Szene dar. Der Hintergrund des Fensters beispielsweise ist einfach ein Objekt mit nur zwei Dreiecken (Quad). Beim Erstellen eines Objekted3ds kann entweder ein vorhandenes Primitiv (Triangle, Quad, Cube, Sphere, Cylinder) verwendet, ein Modell im Format .obj (z.B. aus Blender) geladen oder ein eigenes definiert werden. Jedes Modell besitzt dabei Vertices (Eckpunkte), UV-Koordinaten für die Verteilung der Vertices auf die Textur und Normale (die nach außen Richtung des Vertices). Sollte ein Modell keine Normalen besitzen, ist nur der Unlit-Shader möglich und wird dementsprechend automatisch beim rendern angepasst.

Zum erstellen eines eines Modelles aus eine Prefab kann diese dem Konstruktor übergeben werden:

lwar::Object3d cube = lwar::Object3d(lwar::Primitves::Cube);

Um ein Modell zu laden, muss lediglich der Pfad + Name des Modells als String dem Konstruktor übergeben werden:

lwar::Object3d monkey = lwar::Object3d("monkey.obj");

### Material

Die Klasse Material definiert das Aussehen von Objekten. Hierbei kann der ShaderTyp (Bleuchtet (Standard) oder Unlti), der DrawMode (Points, Lines, Triangels oder Quads) sowie die Textur des Objektes eingestellt werden. Jedes Material hat eine Default-Textur (komplett weiß). Texturen selbst sind alle cv::Mat aus OpenCV. Dadurch können ohne weiteres einfach einfarbige Texturen erstellt (z.B. material.texture = cv::Scalar(112, 25, 25)) oder das Bild der Webcam als Textur verwendet werden. Es können auch Dateien mit dem OpenCV-Befehl imread() geladen werden. Jedes Objekt hat genau 1 Material.

### Transform

Das struct Transform legt die Translation, Rotation und Skalierung eines Modelles im 3D-Raum fest. Translation und Skalierung sind als Vektor, Rotation als Quaternionen umgesetzt. Quaternionen haben den Vorteil, dass es kein Gimbal-Lock beim hintereinander ausführen von bestimmten Drehungen geben kann. Jedes Object3d hat genau 1 Transform.

### Text

In das Framework wurde ein simples Text-Rendering mit Hilfe eines Bitmap-Fonts implementiert (alle zu verwendenden Buchstaben sind in einer Textur mit fester Größe gespeichert). Aktuell sind nur Texte im 2D-Screenspace möglich. Es kann der Text, die Größe und Position festgelegt werden. Um Text der Szene hinzuzufügen, wird die Funktion addText() in der Window-Klasse verwendet.

### Weitere Klassen

Es gibt auch noch weitere Klassen, die allerdings nur Intern verwendet werden und von außen nicht beachtet werden müssen. Diese sind die abstrakte Klasse Renderer und die Implantierung davon OpenGLRenderer, welche das eigentliche Rendering des Bildes auf der Grafikkarte übernehmen. Für OpenGL wird hierbei das Window-Framework GLFW3 eingesetzt, welches sich auch um die Benutzereingaben mit der Tastatur kümmert. Das Framework ist so aufgesetzt, dass man auch noch weitere Renderer implementieren kann (z.B. Vulkan) und dies im Window einfach einstellen kann, ohne das weitere Klassen angepasst werden müssen. Die Klasse OpenGLShader enthält den GLSL-Shader Code für OpenGL als String.

## Render Ablauf

Die folgenden Abläufe sind in der eigentlichen Anwendung nicht zu sehen, werden aber der Vollständigkeit und dem Verständnis des Frameworks kurz erläutert.

Solange die Anwendung läuft, läuft eine Dauerschleife (der Mainloop), welcher bei jedem Durchgang die im Window abgebildete Szene neu rendert. Der OpenGL Renderer verwendet dabei ein doppeltes Puffer-Rendering. D.h. ein Puffer (front buffer) wird auf dem Bildschirm angezeigt, beim anderen wird die Szene in den Puffer gerendert. Nach jedem Durchgang werden die Puffer getauscht, damit die fertige Szene auf dem Bildschirm angezeigt wird. Dieses Verfahren gewährleistet eine flackerfreie, kontinuierliche Bildfrequenz.

Um die Szene im Renderer vorzubereiten, werden vor dem eigentlichen Rendern einige Puffer geleert ( z.B. Tiefenpuffer), dies geschieht in der preDraw() Methode. Das Fenster ruft nun zuerst für evtl. Vorhandene Schrift, dann für jedes Modell der Szene einen Draw-Befehlt des Renderers auf, um das jeweilige Objekt dazustellen. Der Renderer berechnet dafür vor dem Rendern zusätzlich noch die eigentliche Position des Objektes. Für diese Berechnung wird der Transform (Translation, Rotation, Skalierung) des Modells in homogene Koordinaten gebracht (ModelMatrix) und mit der View (Matrix der Kamera, da die Kamera fest ist, in unserem Fall die Einheitsmatrix) - und ProjectionMatrix (setzt sich aus dem Blickwinkel, dem Seitenverhältnis und der near und far plane zusammen) multipliziert (M\*V\*P), die Position selbst wird dann erst in der Grafikkarte für jedem Vertex festgelegt.

Im Abschluss werden dann in der Funktion postDraw() die Puffer getauscht und evtl. Eingabe-Events der Tastatur abgefragt.

## Projekt einrichten

Rechts klick auf den Ordner Projects -> Add New Project

View -> Other Windows -> Property Manager (Eigenschaften Manager)

Je Nach Projekt (32 oder 64) sowohl für Debug, als auch Release ein bestehendes Property Sheet hinzufügen. Dieses befindet sich im Hauptordner des LwAR Ordners unter dem Ordner Templates. Hier dann das passende (32 oder 64) auswählen. Das Template sollte dann bei den Konfigurationen auftauchen. Zusätzlich zum Template muss noch eine Reference zum eigentlichen Framework (LwAR) hinzugefügt werden. Hierzu muss im Solution Explorer im aktuellen Projekt unter References via Rechtsklick -> *Add References* das Projekt LwAR ausgewählt werden.

Die Datei main.cpp im Ordner Template enthält ein Grundgerüst für ein Programm, welches in die bestehende (oder eine neue, falls mein ein leeres Projekt erstellt hat) Hauptdatei kopiert werden kann.

Das Programm sollte nun ohne weitere Änderungen laufen. Da die DLLs nicht installiert wurde, sondern sich nur im Projekt befinden, wurde die Ausgabe .exe Dateien so eingestellt, dass alle im selben Ordner unter Build\x64(x86)\Debug bzw. Release befinden. Hier sind alle notwendigen DLLs enthalten, sodass das Beispielprojekt nun direkt ausgeführt werden kann. Sofern eine Webcam angeschlossen und zur Verfügung steht, sollte sich nun ein neues Fenster mit dem Bild der Kamera öffnen. Das Bild ist an der Y-Achse gespiegelt, damit es wie ein Spiegel wirkt. Das Fenster wird mit dem Drücken der Escape-Taste wieder geschlossen.

## Beispiel Projekte

Das Framework enthält vier Beispielprojekte, welche alle mit dem Framework und beigefügten anderen Frameworks und Bibliotheken umgesetzt wurden, um potenzielle Möglichkeiten des Frameworks zu zeigen. Der folgende Abschnitt soll die Funktion dieser kurz erläutern. Alle Beispielprojekte erstellen ein neues LWAR-Window und ändert jeden Frame den Hintergrund auf das Kamera-Bild ab.

### LwARTest

Dieses Test Projekt zeigt einen Würfel und Text im Hauptfenster ein. Zusätzlich wird noch ein zweites OpenCV Fenster erzeugt, welches ein Bild mit detektierten Kanten der Szene darstellt. Zusätzlich kann der Würfel mit den Pfeiltasten in der Szene verschoben werden.

### LwARTest\_DetectFace

Hier wird mit Hilfe von OpenCV und mit Hilfe der CascadeClassifier Klasse ein Gesicht in der Szene gesucht. Falls eins gefunden wurde, wird ein zuvor eingebunden Modell (Affenkopf aus Blender) auf die Position des Kopfes verschoben.

### LwARTest\_DetectRedCircles

Bei diesem Projekt werden mit Hilfe der OpenCV Funktion HoughCircles und der vorherigen Filterung auf eine rote Farbe Rote Kreise in der Szene gesucht. Für jeden gefundenen Kreis wird ein Würfel erstellt und auf die Position des Kreises gebracht. Die Würfel drehen sich noch zusätzlich jeden Frame um Ihre eigene Achsen.

### LwARTest\_Marker

Hier wird die freie Marker Bibliothek Arucu genutzt, um einen Marker von Arucu in der Szene zu finden. Auf diesem Marker wir dann ein Wüfel plaziert. Die Marker können beispielsweise unter <http://terpconnect.umd.edu/~jwelsh12/enes100/markergen.html> generiert werden.

Resultate

Im Rahmen dieser Projektarbeit wurde ein Framework für Augmented Reality Anwendungen in OpenGL und OpenCV implementiert. Alle notwenidgen Bibliotheken im Framework sind vorcompiliert enthalten. Zusätzlich wurde alle notwendigen Einstellungen in externen Property Templates implementiert, sodass mit wenig Aufwand mit Hilfe des Frameworks eine funktionierende Anwendung

Diskussion

<Wie sind die Ergebnisse zu interpretieren, welche Fragestellungen ergeben sich daraus, wie sehen die Ergebnisse im Vergleich zu anderen Publikationen aus.>

Fazit und Ausblick

Wie bei allen Projekte gibt es hier beim Framework auch noch Erweiterungsmöglichkeiten, welche im folgenden Abschnitt vorgestellt werden sollen.

Der Renderer wurde so implantiert, dass noch zusätzliche Render-Backends implementiert werden können. So könnte beispielsweise noch den OpenGL Nachfolger Vulkan implementieren.

Das Framework läuft zurzeit nur auf Windows mit Visual Studio, denkbar wäre noch, weitere Plattformen wie z.B. Linux oder sogar Android (via NDK).

Das Rendern von Texten ist nur sehr einfach implementiert. Falls bessere Möglichkeiten zur Texterstellung notwendig sind, könnte man eine externe Bibliothek, z.B. Freetype hinzufügen.

ArbeitsbuchReferenzen

<Verwendete Publikationen, Materialien, ...  
Bitte bei allen Materialien, auch bei eigenen (z.B. Fotos), immer die jeweilige Quelle angeben. In den Anhängen können sie dafür das Kommentarfeld verwenden.>

Ressourcen

<ggf. Link auf gepacktes Archiv der Arbeitsumgebung/Quelltexte/verwendete Libraries.>