Audio-Video-Programmierung

Prof. Dr. Plaß & Sudau

WiSe 17/18

https://github.com/CarinaKr/SoundSpace

SoundSpace

Takes you wherever you want to be.

Carina E. Krafft | 2269579
Jordanis Lazaridis | 2142772
Irena Becker | 2238833
Kevin Hagen | 2270985

23.01.2018
Hochschule für angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Inhalt

Einleitung

„SoundSpace“ ist ein tangible-Sound Projekt, bei dem es dem Nutzer möglich ist durch Objekte verschiedener Formen und Farben seine eigene Soundkulisse zu erzeugen. Dafür können Würfel oder Rechtecke, Kugeln oder Kreise und dreiseitige Pyramiden oder Dreiecke, jeweils in den Farben Rot, Blau und Grün, frei in einem kalibrierten Bereich platziert werden. Jede Form steht hierbei für einen eigenen Soundclip und jede Farbe für eine bestimmte Lautstärke. Über das Webinterface kann der User das Setup weiter konfigurieren. Die verschiedenen Formen gibt es in zwei- sowie dreidimensionaler Form. Letztere sind eine Erweiterung des ursprünglich in 2D durchgeführten Projekts. Wo im Weiteren beispielsweise von Würfeln gesprochen wird, könnte genauso gut ein Rechteck genutzt werden.

Das Projekt „SoundSpace“ wurde im Rahmen der Vorlesung Audio-Video-Programmierung realisiert. Eine Projektvoraussetzung war daher die Verwendung der in der Vorlesung behandelten Programmiersprachen. Dies sind C++, unter Verwendung der Entwicklungsumgebung QT sowie der Bibliothek openCV, und JavaScript in Kombination mit der WebAudio API.

Die Entwicklung des Projektes wurde grundlegend in die Bereiche Backend-, Frontend- und Schnittstellenentwicklung unterteilt, sodass an allen Bereichen unabhängig voneinander gearbeitet werden konnte.

Backend Entwicklung

Das Backend von „SoundSpace“ wurde in C++ entwickelt. Ziel des Backends ist es ein eingespeistes Kamerabild zu analysieren. Die Analyse ermittelt zu jedem einzelnen Objekt jeweils Farbe, Form und Position im dreidimensionalen Raum. Diese Daten werden dann per MIDI-Schnittstelle ans Frontend weitergeleitet.

Formerkennung

Eine unserer wichtigsten Anforderungen ist die Unterscheidung mehrerer Objekte gleicher Farbe und Form. Als Lösung dieses Problems wird die Erkennung und Unterscheidung der Objekte über die Form geregelt. Damit dies problemlos läuft, müssen im zu analysierenden Bild möglichst eindeutige Formen erkennbar sein. Aus diesem Grund filmt die Kamera aus der Vogelperspektive. Als Objekte haben wir uns für eine Kugel, eine dreiseitige Pyramide und einen Würfel entschieden.

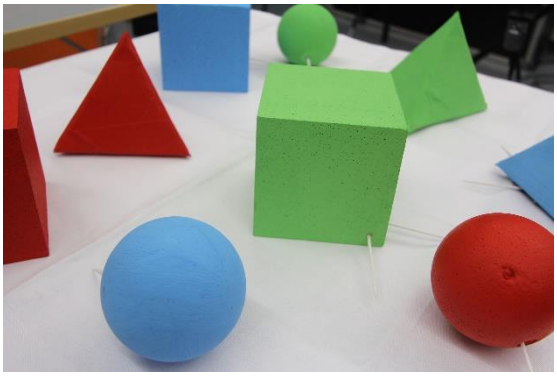

Abbildung 1: 3D Objekte in Frontansicht

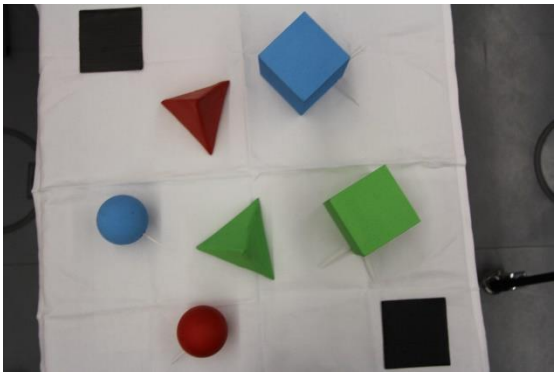

Abbildung 2: 3D Objekte in Vogelperspektive

Für die Formerkennung wurde die Klasse ShapeRecognition geschrieben, welche von VideoProcessor erbt. Die process-Methode nimmt den aktuellen Kameraframe entgegen und erkennt in mehreren Schritten die Form, absolute Pixelkoordinaten und absolute Größe jedes Objektes. Als erstes wird dafür der Kameraframe in ein schwarz-weiß Bild umgewandelt. In diesem Schritt ist es wichtig die Schatten der Objekte herauszufiltern, um die Form nicht zu verfälschen. Da in unserer Anwendung

für die Kalibrierung die Farbe Schwarz verwendet wird, werden die Schatten jedoch erst nach der Kalibrierung herausgefiltert. Aus dem so entstandenen schwarz-weiß Bild werden im zweiten Schritt mithilfe der findContours- und approxPolyDP – Methoden, genauere Informationen über die Konturen der einzelnen Objekte, sowie deren Position und Größe, gewonnen. Durch Betrachtung der Seitenanzahl und Winkelbestimmung kann zwischen einem Rechteck, Dreieck und Kreis unterschieden werden. Die Winkelbestimmung ist primär deshalb notwendig, da die dreidimensionalen Figuren bei ungünstiger Position zu der Kamera verfälschte Konturen aufweisen können und so eine dreieckige Figur manchmal vier Seiten hat – da die Winkel jedoch nicht stimmen, kann so der Fehler identifiziert werden.

Die ermittelten Daten werden in ein Array von structs gespeichert. Das struct objData hält Informationen über die Position, Größe, Form und Farbe des Objektes. Wir nutzen es als Datencontainer für unsere erkannten Figuren.

Farberkennung

Nach der Bestimmung der Formen und ihrer Position, widmen wir uns ihrer Farbe. Dafür nutzen wir die Klasse ColorProcessor, welche bereits im Unterricht behandelt wurde. Sie wurde für unsere Zwecke ein wenig erweitert und modifiziert.

Auch die Klasse ColorProcessor erbt von VideoProcessor. Da wir bereits wissen an welchen Positionen sich Objekte befinden, wissen wir ebenfalls wo wir nach Farben suchen müssen. So kann in der process-Methode einfach über alle gefundenen Objekte iteriert, und damit viel Leistung im ColorProcessor gespart werden. Per statischen Zugriff holen wir uns die jeweilige absolute Position des Objektes. Mit der Methode getHSVAtPoint(int x, int y) ermitteln wir dann den Farbwert an der gegebenen Position. Dieser wird in die Methode checkForColor(int hueToCheck, int satToCheck, int valToCheck) gegeben. Die Methode prüft, auf welche der vordefinierten Farbwerte der HSV-Farbwert zutrifft. Sie vergleicht ihre Parameter mit von uns vordefinierten Farbwerten, welche den Farben Rot Grün und Blau entsprechen. Außerdem wird vorab einmal mit der Farbe schwarz verglichen, welche wir als „CALIBRATION_COLOR“ nutzen. Trifft keine der definierten Farben zu, handelt es sich um eine „UNKNOWN_COLOR“. In dem struct für das jeweilige Objekt wird der Farbwert als enum hinterlegt.

Alle Objekte die eine „UNKNOWN_COLOR“ besitzen, werden gelöscht. Nach der Kalibrierung löschen wir ebenfalls alle Objekte mit der „CALIBRATION_COLOR“. Diese Objekte sollen nicht ans Frontend geschickt werden, da sie keine Soundquellen darstellen und daher im Frontend nicht benötigt werden.

Kalibrierung und Positionsbestimmung im dreidimensionalen Raum

Weil wir uns nicht auf bestimmte Maße beschränken wollten, haben wir uns überlegt eine Raumkalibrierung einzuführen. Dadurch kann der Nutzer sich seinen Raum so gestalten wie er möchte, er muss ihn nur zu Beginn kalibrieren. Dieser Prozess geschieht in der Calibration-Klasse.

Hierfür platziert der Anwender je ein schwarzes Rechteck, an zwei sich diagonal gegenüberliegenden Punkten des Raumes. Sobald er mit der Ausrichtung zufrieden ist, kann die Kalibrierung per Button im UI gestartet werden. Während der Kalibrierung sucht das Programm über einen bestimmten Zeitraum gezielt nach zwei schwarzen Rechtecken. Sobald diese gefunden werden, wird aus ihren Koordinaten die obere, linke Ecke des Raumes als Punkt (0, 0), die Größe des Raumes, sowie sein Mittelpunkt berechnet. Die Größe der Rechtecke wird als Standardgröße festgelegt. Aus ihr berechnen wir die Brennweite der Kamera, was später für die Bestimmung der Z-Achse der Figuren notwendig ist. Wichtig ist zu beachten, dass hierbei mit relativen Werten und nicht mit der realen Größe der Objekte gerechnet wird. Die errechnete Brennweite ist also ein fiktiver Wert und stimmt nicht mit der tatsächlichen Brennweite der verwendeten Kamera überein.

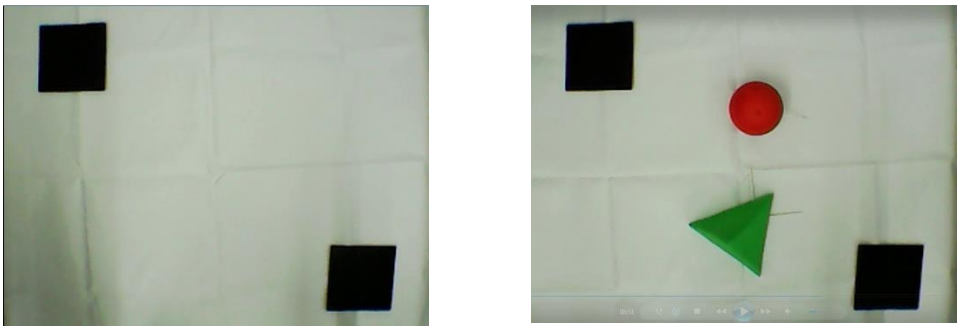

Abbildung 3: Links: Raum zwischen den Kalibrierungsrechtecken. Rechts: Kalibrierter Raum kann nun mit Objekten versehen werden

Bei der Bestimmung der Position im kalibrierten Raum gilt es, die absoluten Koordinaten die wir durch die Formerkennung erlangen in relative Raumkoordinaten umzurechnen. Diese befinden sich im Wertebereich von 0-100%, um so einen möglichst generischen Raum simulieren zu können. Die Berechnung der Z-Koordinate der Objekte erfolgt auf eine andere Art und Weise: Hierfür wird zuerst über die Brennweite und Standardgröße der Abstand des Objektes zur Kamera berechnet. Anschließend wird mithilfe der X- und Y-Koordinaten über eine Dreiecksberechnung die Z-Position bestimmt.

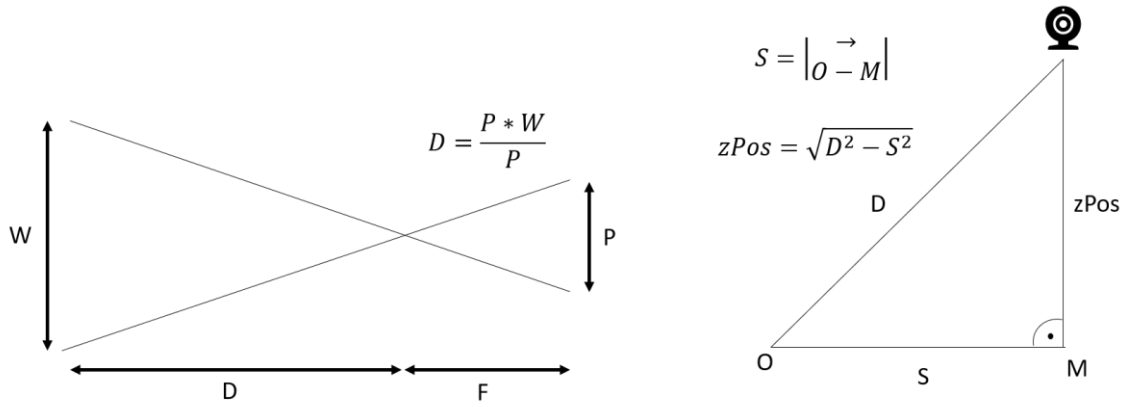

Abbildung 4: Bestimmung der Z-Koordinate eines Objektes im dreidimensionalen Raum

Frontend Entwicklung

Das UI

Das intuitive Webinterface ermöglicht es dem Anwender verschiedene Einstellungen anhand von Karten zu ändern. Für jedes unterschiedliche Objekt gibt es genau eine Karte, über die Lautstärke und Soundclip konfigurierbar sind. Der User kann also beispielsweise den Soundclip für alle roten Rechtecke austauschen und die Lautstärke um eine gewünschte dB-Anzahl verstärken. Weiterhin gibt es verschiedene Themen, die erlauben unterschiedliche Soundkulissen zu bauen. Zur Laufzeit kann der Anwender das Thema austauschen und sich somit zum Beispiel mühelos von der Stadt in den Wald teleportieren.

Damit der Anwender immer einen genauen Überblick hat, was gerade passiert, filmt eine Webcam den repräsentierten Raum und das Bild wird ans Frontend übertragen. Neben dem Kamerabild werden nochmals alle sich im Raum befindlichen Objekte hierarchisch dargestellt. In dieser Darstellung hat der Nutzer die Möglichkeit die Lautstärke der einzelnen Objekte zu konfigurieren.

Anwendung der WebAudio-API und Resonance Audio SDK

Um den Audioteil zu realisieren haben wir die WebAudio-API und die Resonance Audio SDK genutzt. Die Resonance Audio SDK ist eine hilfreiche Bibliothek um schnell und einfach 3D-Sound zu erzeugen.

Zu Beginn der Anwendung werden jeweils 20 HTML-Audio-Elemente, MediaElementSources, Gain-Nodes und ResonanceAudio Soundquellen erzeugt. Diese Elemente werden in genannter Reihenfolge verbunden. Wenn zur Laufzeit Daten im Frontend ankommen, wertet dieses Form und Farbe des Objektes aus und erkennt die momentane Standardeinstellung des entsprechenden Objektes. Aus der Standardeinstellung werden daraufhin Lautstärke und Soundclip übernommen – die Anpassung dieser Attribute erfolgt dann jeweils an der Gain-Node bzw. der HTML-Audio-Source. Durch die Schaltung der Audio-Nodes können wir so einfach und dynamisch zur Laufzeit die Soundquellen austauschen, ohne jedes Mal Elemente zu erzeugen und zu löschen.

Um die Standardeinstellungen und die Einstellungen an den aktuellen Objekten zu speichern, nutzen wir JavaScripts Prototype-Objekte: Wir haben eine Klasse „Soundobject“ programmiert, die alle relevanten Informationen als Datencontainer hält und Methoden hat um die Lautstärke und den Soundclip zu aktualisieren.

Anfangs initialisieren wir für jede reale Figur genau ein defaultSoundObject, mit eigenen Einstellungen. Außerdem initialisieren wir ebenfalls 20 leere currentSoundObjectsInScene. Letztere werden später genutzt um den virtuellen Raum zu füllen.

Legt der User nun ein Objekt in den Raum und das Backend schickt diese Daten ans Frontend, kommen die Daten als JSON an. Dieses JSON übersetzen wir und speichern die darin enthaltenen Informationen in dem nächsten freien currentSoundObjectInScene. Nachdem diese Daten gespeichert wurden, wird geprüft zu welchem der defaultSoundObjects das Objekt passt – m.a.W. vergleichen wir Form und Farbe des neuen Objekts mit den bekannten Standardobjekten. Sobald ein passendes gefunden wird, übernimmt es von diesem die letzten fehlenden Einstellungen (Soundclip und Lautstärke). Abschließend werden die korrespondierende HTML-Audio-Source und die Resonance Audio Source aktualisiert. Dadurch wird dann an entsprechender Position im Raum der gewünschte Ton erzeugt.

Die Resonance Audio SDK erlaubt es schnell und einfach einen virtuellen, dreidimensionalen Raum zu erzeugen. Man kann zudem Materialien für die Wände angeben, die den Hall verändern und den Raum seinen Wünschen nach bemaßen. Anschließend kann man mit resonanceAudioScene.createSource() eine neue 3D-Soundquelle erzeugen und sie im Raum positionieren. Dies geschieht über den Aufruf von source.setPosition(width, height, depth). In unserem Quelltext nutzen wir dies alles in einem weiteren Prototype-Objekt namens threeDAudio. Hier können wir die 3D-Szene direkt so konfigurieren wie wir wünschen und in eigenen Methoden auch die Positionsangaben in die für Resonance Audio erforderlichen Werte umrechnen.

Schnittstellenprogrammierung

Zur Übertragung der Daten vom Backend ans Frontend haben wir uns für die im Unterricht behandelte MIDI-Schnittstelle entschieden, in Kombination mit dem virtuellen MIDI-Driver „LoopBe1“. Dabei haben wir uns allerdings, anders als in der Vorlesung, dazu entschieden die Daten

in Form von MIDI System Exclusives zu kodieren und zu übertragen. Grund hierfür ist, dass die zu verschickende Datenmengen zu groß für die Standardbefehle ist.

```
data[0] = 0xf0;                              //start byte
data[1] = (objectShape << 4) | (objectColor); //first 4 bit: shape; second 4 bit: color
data[2] = relativePosition.x;                //xPos
data[3] = relativePosition.y;                //yPos
data[4] = zPos;                              //zPos
data[5] = 0xf7;                              //end byte
```

Abbildung 5: Verschlüsselung der Kalibrierungsdaten

Um etwas Performance zu sparen schicken wir nur alle 24 Frames die Daten ans Frontend. 24 Frames entsprechen dabei etwa einer Sekunde, einem Intervall das sich bewährt hat. Über diese 24 Frames bilden wir zudem einen Durchschnitt. Dafür werden alle Objekte gesammelt, verglichen und dies abschließend ausgewertet. Es wird angenommen, dass Objekte sich nur über eine bestimmte Distanz in diesem Zeitraum bewegen können. Objekte annähernd gleicher Position werden also einander zugeordnet und es wird angenommen, dass es sich dabei um das gleiche Objekt handelt. Nur jene Objekte, die zuverlässig über den Verlauf der 24 Frames erkannt wurden, werden ans Frontend weitergeleitet. Somit können wir Fehler im Form- und Farberkennungsalgorithmus abfangen und ausgleichen.

Fazit

Wir sind mit dem Projekt „SoundSpace“ sehr zufrieden. Auf der technischen Seite ist es den meisten Anforderungen gerecht geworden. Wir konnten fast all unsere User Stories abarbeiten, mit Ausnahme der Umsetzung auf der 3D-Soundanlage im Tonstudio der Finkenau. Dies liegt an Kommunikationsproblem zwischen der Resonance Audio SDK und Max MSP ist daher ein annehmbarer Rückschlag für uns. Zudem konnte das Projekt nichtsdestotrotz auf 5.1 Kopfhörern durchgeführt werden, wobei allerdings einige Klanginformationen verloren gehen.

Es gab auch einige technische Herausforderungen. So war beispielsweise das Entwerfen einer responsiven Anwendung schwieriger und zeitaufwändiger als gedacht. Vor allem die Wahl der richtigen Objekte und Farben war eine Schwierigkeit. Wir haben festgestellt, dass Lichtverhältnisse und Betrachtungswinkel viel Einfluss haben, aber es hat uns Spaß gemacht mit diesen Elementen zu experimentieren.

Im Laufe der Entwicklung haben wir uns einige Gedanken über die Erweiterbarkeit des Projektes gemacht. Neben der Möglichkeit neue Objekte und Farben erkennen zu können, wäre eine mögliche Erweiterung eine Suchfunktion in der Website zu integrieren. Diese Suchfunktion wurde im Layout des Frontend-UI bereits umgesetzt, erfüllt derweilen allerdings noch keine Funktion.

Abschließend lässt sich sagen, dass „SoundSpace“ ein interessantes Projekt mit vielen technischen Herausforderungen war, bei dem wir viel gelernt haben. Es war ein schöner Einstieg in die Programmiersprache C++ und die WebAudio-API. Außerdem konnten wir bereits vorhandenes Wissen festigen und uns mit interessanten Algorithmen, wie zum Beispiel der Formerkennung, beschäftigen.

UML-Diagramme

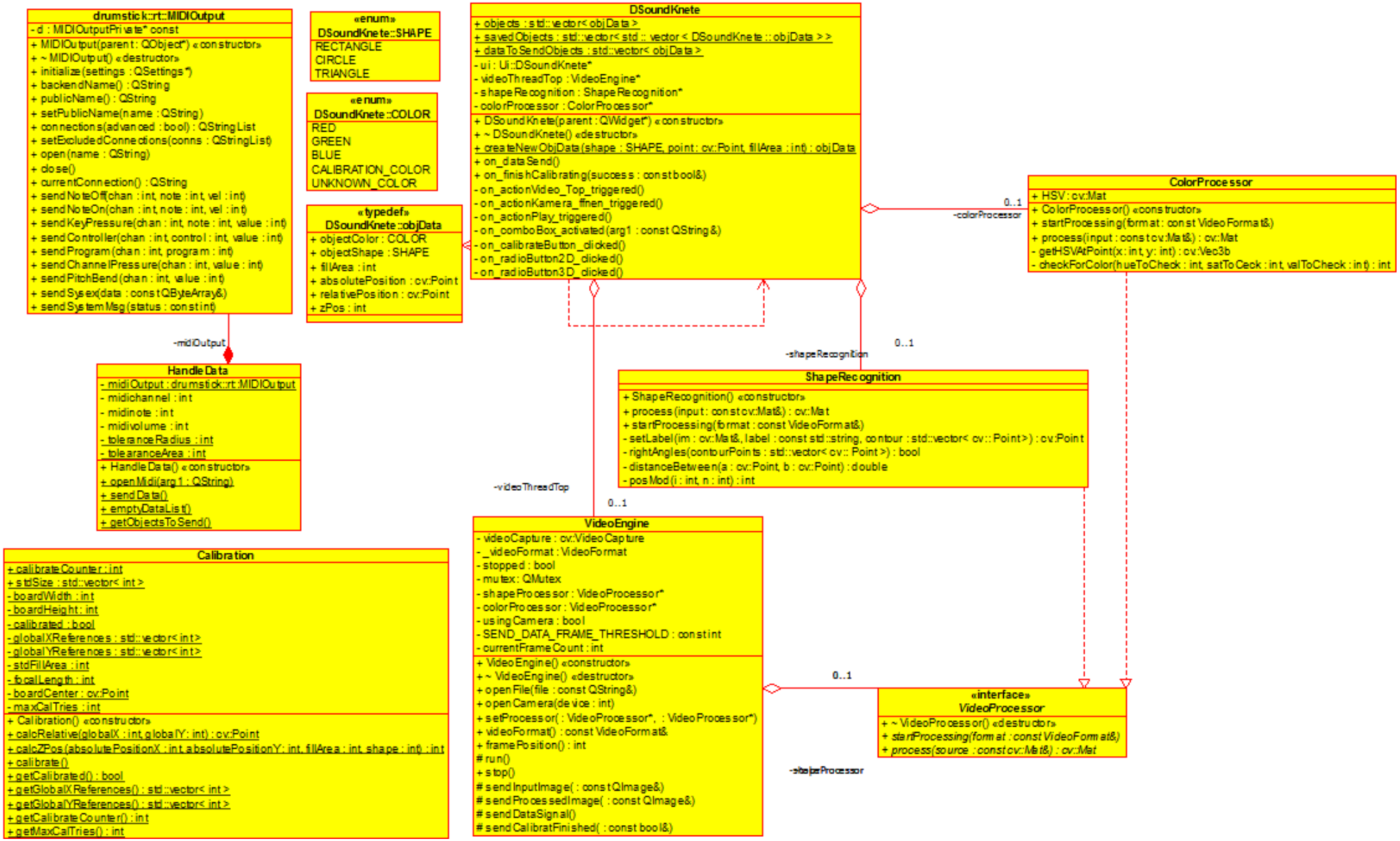

Abbildung 6: UML-Klassendiagramm des Backends (weniger relevante Klassen entfallen der Einfachheit halber)

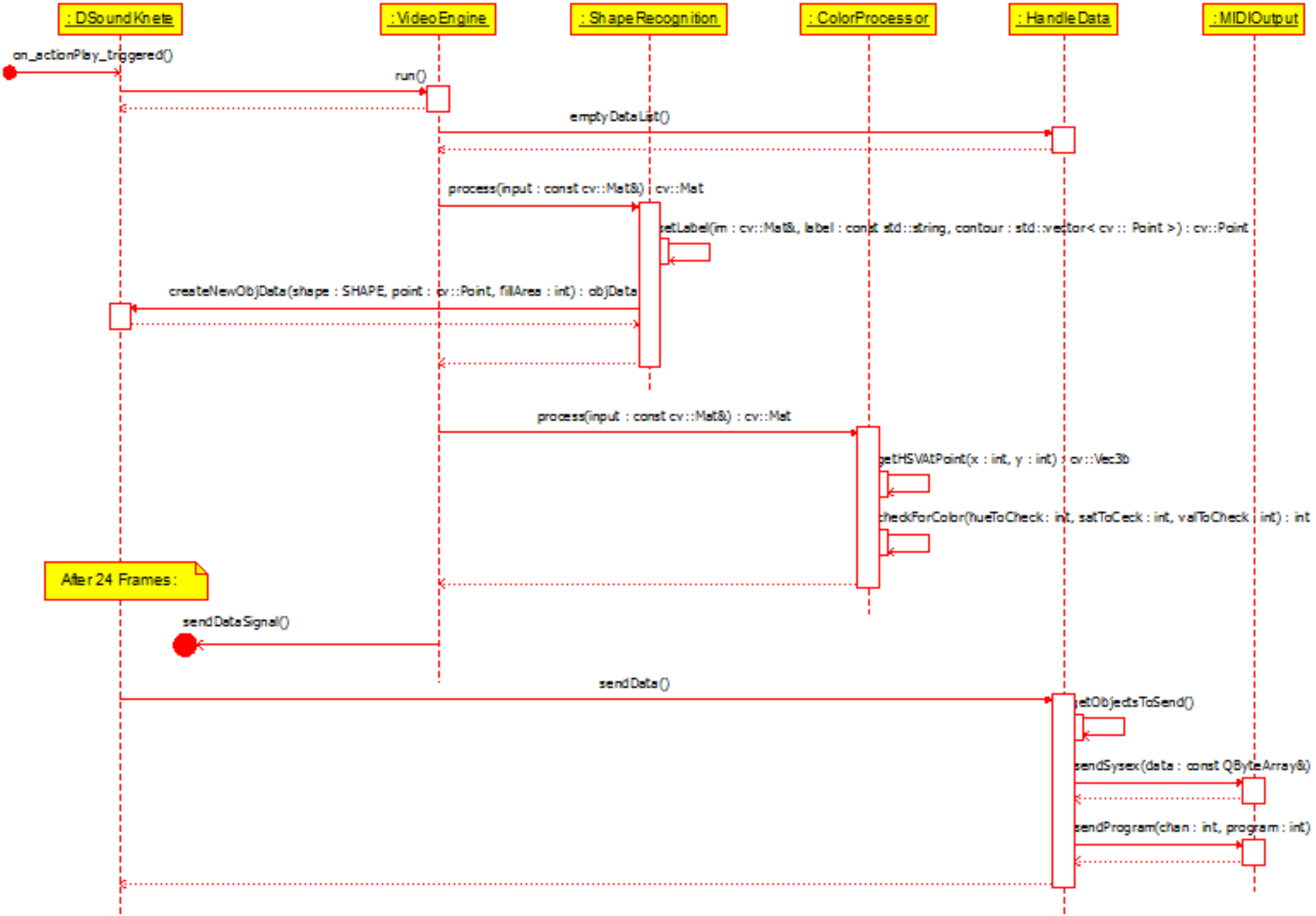

Abbildung 7: UML-Sequenzdiagramm des Backend (weniger relevante Klassen und Methoden entfallen der Einfachheit halber)