

# Dokumentation VideTechTools

---

*VideoTechTools* ist eine Sammlung von Simulationen für die Videotechnik-Vorlesung.

## Installation

Zur Entwicklung der App wird zunächst die Anwendung Expo-CLI benötigt:

```
yarn add expo-cli
```

Daraufhin müssen im Haupt-Verzeichnis von VideoTechTools mit folgender Anweisung alle verwendeten Bibliotheken installiert werden:

```
yarn install
```

Falls Node.JS nicht automatisch mitinstalliert wird, kann dies hierüber erfolgen: [Node.JS](#)

## Aufbau der Software

```
VideoTechTools
├── README.md
├── App.js      // Einstiegspunkt der App
├── app.json    // Metadaten etc. der App
├── ...
└── src
    ├── calculations // Mathematische Funktionen/Umrechnungen
    ├── components   // Wiederverwendbare Komponenten
    ├── navigation   // Alles zu Navigationen
    └── screens      // Aus components zusammengesetzte Seiten
    ...
...
```

# Signal-Generator

---

## SignalGeneratorView

```
export const SignalGeneratorView = (
  {
    setSignal,
    setEncodingVideoStandard,
    setEncodingBitDepthIdx,
    showHideButton = false,
    style=undefined
  }) => {...}
```

**setSignal:** Signal-Ausgang: Übergibt ein signalYCBCR siehe [Signal Arrays](#) über diesen Parameter an Eltern-Objekt.

**setEncodingVideoStandard:** Index des Videostandards, dem signalYCBCR entspricht ([0..2] für Rec.601, 709 oder 2020)

**setEncodingBitDepthIdx:** Index des Quantisierungsgrads, dem signalYCBCR entspricht ([0..2] für 8-, 10- oder 12-Bit)

**showHideButton:** Button zum Ausblenden des Generators einblenden (true/false)

**style:** Zum annehmen von Style-Parametern

- Errechnet ein signalYCBCR aus einem signalSmallRGB.
- Buttons zum Wechsel des Videostandards und Quantisierung des signalYCBCR
- Beinhaltet einen GeneratorContainer um das signalSmallRGB zu beschreiben.
- Beinhaltet Corrector um das signalSmallRGB zu verändern.

## Optimierungen

- Benutzeroberfläche überarbeiten und Platz besser ausnutzen, damit die ScrollView weg kann.
- Wenn ScrollView weg ist und Performance optimiert wurde können Slider zum Einstellen der entsprechenden Parameter verwendet werden.
- Zur Vereinheitlichung den Parameter "setSignal" in "setSignalYCBCR" und "setEncodingVideoStandard" in "setEncodingVidStdIdx" umbenennen (und entsprechende Parameter in den Screens). Aus unerklärlichen Gründen führt dies zu dem Fehler "setSignalYCBCR is not a function.".

Verwendete GeneralComponents:

--

---

## Subkomponenten

---

### GeneratorContainer

```
const GeneratorContainer = (
  {
    setSignalSmallRGB,
    generatorIdx,
    setGeneratorIdx
  }) => {...}
```

**setSignalSmallRGB:** Signal-Ausgang: Übergibt ein signalSmallRGB (siehe [Signal Arrays](#)) über diesen Parameter an Eltern-Objekt.

**generatorIdx:** Index des Generators, der verwendet und angezeigt werden soll.

**setGeneratorIdx:** Veränderung des Generator-Index beim Eltern-Objekt.

- Sammelt alle Generator-Typen und stellt Schaltflächen zum Wechsel des anzuzeigenden Generators bereit.

### Hinweise

--

### Optimierungen

- Ggf. mittels "[Containment](#)" umsetzen, um alles kompakter und flexibler für neue Signalarten zu machen. Gleiches Prinzip wie mit den Settings-Objekten.
-

## FullColorGenerator

```
export const FullColorGenerator = ({ setSignalSmallRGB })=>
{...}
```

**setSignalSmallRGB:** Signal-Ausgang: Übergibt ein signalSmallRGB (siehe [Signal Arrays](#)) über diesen Parameter an Eltern-Objekt.

- Erzeugt ein einfarbiges Vollbild als signalSmallRGB.
- Benutzeroberfläche mit RGB- und HSV-Modus.

### Hinweise

- showingRgbControls dient neben dem Wechsel der Anzeige vor Allem dazu, dass sich die useLayoutEffect()-Hooks nicht in einer Endlosschleife verfangen.

### Optimierungen

- TapButtons durch Slider ersetzen wenn Optimierungen in [SignalGeneratorView](#) erfüllt sind.
- Option um Überpegel mit den RGB-Parametern zu erzeugen.

---

## GradientGenerator

```
export const GradientGenerator = ({ setSignalSmallRGB })=>
{...}
```

**setSignalSmallRGB:** Signal-Ausgang: Übergibt ein signalSmallRGB siehe [Signal Arrays](#) über diesen Parameter an Eltern-Objekt.

- Erzeugt ein Farbverlauf als signalSmallRGB.
- Wechsel von horizontalen und vertikalen Verlauf
- Schnellauswahl wichtiger Farbwerte durch Button-Matrix (ColorPad).

### Hinweise

--

## Optimierungen

- Ggf. zusätzliche Ansicht mit detaillierteren Einstellungsmöglichkeiten der Farbwerte (mit RGB und HSV-Slidern)
- 

## BarsGenerator

```
export const BarsGenerator = ({ setSignalSmallRGB })=> {....}
```

**setSignalSmallRGB:** Signal-Ausgang: Übergibt ein signalSmallRGB siehe [Signal Arrays](#) über diesen Parameter an Eltern-Objekt.

- Erzeugt EBU-Farbbalken als signalSmallRGB.
- Benutzeroberfläche mit Wechsel zwischen 100/100 und 100/75

## Hinweise

--

## Optimierungen

- Weitere Test-Bilder
- 

## Corrector

```
export const Corrector = (
  {
    contrastOffset,
    setContrastOffset,
    gammaOffset,
    setGammaOffset,
    brightnessOffset,
    setBrightnessOffset
  }) => {
```

**contrastOffset:** Aktueller Wert des Kontrast-Versatz-Parameters des Eltern-Objekts.

**setContrastOffset:** Übergibt neuen Wert des Kontrast-Versatz-Parameters ans Eltern-Objekt.

**gammaOffset:** Aktueller Wert des Gamma-Versatz-Parameters des Eltern-Objekts.

**setGammaOffset:** Übergibt neuen Wert des Gamma-Versatz-Parameters ans Eltern-Objekt.

**brightnessOffset:** Aktueller Wert des Helligkeit-Versatz-Parameters des Eltern-Objekts.

**setBrightnessOffset:** Übergibt neuen Wert des Helligkeit-Versatz-Parameters ans Eltern-Objekt.

- Benutzeroberfläche zum Verändern der Corrector-Parameter im Eltern-Objekt.

## Hinweise

--

## Optimierungen

- TapButtons durch Slider ersetzen wenn Optimierungen in [SignalGeneratorView](#) erfüllt sind.
  - Sättigungs-Regler hinzufügen.
-

## Berechnungen

### Komponenten

- CIE-Normfarbtafel
- Vektorskop
- Waveformmonitor
- Signal-Generator
- Signal-Vorschau
- Allgemeine Komponenten
  - Einstellungs-Fenster
  - ScopesCamera
  - VideoStandardAlertView

## Navigation

### Seiten

- CIE-Normfarbtafel
- Waveformmonitor
- Vektorskop
- Komponentensignal
- Scopes Übersicht

### Vorkommende Signal-Arrays

## Nützliche Tutorials

React

React Native

Durchreichen von Daten

Three.JS

React-Three-Fiber

Expo

## Dokumentationen der verwendeten Bibliotheken

[React](#)

[React Native](#)

[Three.JS](#)

[React-Three-Fiber](#)

[Expo](#)

# Calculations

---

## CalcColorSpaceTransform

- Funktionen zum Umrechnen von Farbmodellen.

Für einzelne Tripel:

```
export function cvtRGBtoXYZ(rgb_array, colorSpace = "709") {...}
export function cvtHSVtoRGB(HSV_array) {...}
export function cvtRGBtoHSV(RGB_array) {...}

export function cvtXYZtoxy(XYZ_array) {...}
export function cvtXYZtoxyY(XYZ_array) {...}
```

Für Singal-Arrays:

```
export function cvtSignalRGBtoXYZ(signalRGB, colorSpace = "709") {...}

export function cvtSignalXYZtoxy(signalXYZ) {...}
export function cvtSignalXYZtoxyY(signalXYZ) {...}
```

---

## CalcComponentSignal

- Funktionen zum Rechnen mit Komponentensignalen.

Für einzelne Tripel:

```
export function cvtRGBtoYCBCR(RGB, standard = "709") {...}
export function cvtYCBCRtoRGB(YCBCR, standard = "709") {...}

export function upscaleYCBCR(YCBCR, bitDepth = 10) {...}
export function downscaleYCBCR(YCBCR, bitDepth = 10) {...}
```

```
export function limiterYCBCR(YCBCR, bitDepth, fullVideoData = false) {...}
```

Für Singal-Arrays:

```
export function cvtSignalRGBtoYCBCR(signalRGB, videoStandard = "709" ) {...}
export function cvtSignalYCBCRtoRGB(signalYCBCR, videoStandard = "709" ) {...}

export function upscaleSignalYCBCR(signalSmallYCBCR, bitDepth = 10 ) {...}
export function downscaleSignalYCBCR(signalYCBCR, bitDepth = 10 ) {...}

export function limiterSignalYCBCR(signalYCBCR, bitDepth, fullVideoData = false) {...}
export function limiterSignalSmallRGB(signalSmallRGB, fullVideoData = false) {...}
```

---

## CalcHelpers

- Sonstige Hilfs-Funktionen.

```
export function rgbToString(rgbArray){...}
export function rgbToComplColorString(rgbArray){...}
export function clamp(value, min = 0, max = 1) {...}
```

---

## CalcRGBSignal

- Funktionen des Correctors.

Für einzelne Tripel:

```
export function upscaleRGB(RGB, bitDepth = 10) {...}
export function downscaleRGB(RGB, bitDepth = 10) {...}
```

Für Singal-Arrays:

```
export function upscaleSignalRGB(signalSmallRGB, bitDepth = 10)
) {...}
export function downscaleSignalRGB(signalRGB, bitDepth = 10 )
{...}
```

---

## CalcSignalCorrector

- Funktionen des Correctors.

Für einzelne Tripel:

```
function offsetContrast(pixelValue = [0, 0, 0], m = 1) {...}
function offsetBrightness(pixelValue = [0, 0, 0], b = 0) {...}
function offsetGamma(pixelValue = [0, 0, 0], gamma = 1,
maxValue = 1) {...}
```

Für Singal-Arrays:

```
export function offsetSignalContrast(signalRGB, m = 1) {...}
export function offsetSignalBrightness(signalRGB, b = 0) {...}
export function offsetSignalGamma(signalRGB, gamma = 1,
maxValue = 1) {...}
```

---

## CalcSignalGenerator

- Funktionen der Signal-Generatoren.

```
export function generateRGBSignalFullColor(valueRGB, width, height){...}
export function generateRGBSignalBars(width = 8, height = 1, type100 = true){...}
export function generateRGBSignalGradient(startRGB, endRGB, width, height, directionHorizontal=true){...}

function blendColor(firstRGB, secondRGB, ratio = 0.5){...}
```

---

## CalcHelpers

- Sonstige Hilfs-Funktionen.

```
export function rgbToString(rgbArray){...}

export function rgbToComplColorString(rgbArray){...}

export function clamp(value, min = 0, max = 1) {...}
```

---

# CIE-Normfarbtafel

---

## CieView

```
export const CieView = (
  {
    signalYCBCR,
    withOverlays = false,
    encodedVideoStandard = 1,
    encodedBitDepthIdx = 0
  }) => {...}
```

**signalYCBCR:** siehe [Signal Arrays](#)

**withOverlays:** Buttons und Labels anzeigen (true/false)

**encodedVidStdIdx:** Index des Videostandards, dem signalYCBCR entspricht ([0..2] für Rec.601, 709 oder 2020)

**encodedBitDepthIdx:** Quantisierungsgrad, in dem signalYCBCR vorliegt ([0..2] für 8-, 10- oder 12-Bit)

- Rechnet Signal für Visualisierung um
- Stellt Einstellungs-Menüs für Visualisierung bereit

## Optimierungen

- Einen Weg finden, um [OrbitControls](#) zu integrieren (bisher Problem mit Paketversion von [drei](#))
- Farbmanagement des Canvas für korrekte Farbwiedergabe anpassen

Verwendete GeneralComponents:

[ScopesCamera](#),  
[VideoStandardAlertView](#),  
[SettingsPopOverContainer](#)

---

## Subkomponenten

---

## CiePlot

```
const CiePlot = (
  {
    signalxyY,
    signalRGB,
    dotSize = 0.01
  }) => {...}
```

**signalxyY, signalRGB:** siehe [Signal Arrays](#)

**dotSize:** Radius der Punkte im Plot

- Bringt Bildpunkte des signalxyY in eine CIE-Normfarbtafel mit den entsprechenden Farben aus dem signalRGB

### Hinweise

- Kann nur innerhalb eines React-Three-Fiber Canvas verwendet werden

### Optimierungen

- Alternativ zu der aktuell relativen Darstellung der Farben: Modus um "echte Farben" darzustellen, sodass (durch sRGB-Farbraum-Begrenzung aus React Native) alle Farben außerhalb sRGB immer voll gesättigt dargestellt werden.
- Hier lässt sich die Performance noch deutlich optimieren, indem systematischer mit den Hooks "useMemo()", "useRef()",... gearbeitet wird und unter Verwendung eines [InstancedMesh](#). Hier muss aber auf die Paketversion von Three geachtet werden, da neuere Versionen mit anderen Bibliotheken (wie react-three-fiber) Probleme bereiten kann.

---

## COS

```
export const COS = (props) => {...}
```

**props:** --

- Koordinatensystem

## Hinweise

- Kann nur innerhalb eines React-Three-Fiber Canvas verwendet werden

## Optimierungen

- Achsenbeschriftungen fehlen
  - Achsen dürfen nicht bunt sein
- 

## CieBounds

```
export const CieBounds = () => {
```

### props: --

- Reine Spektrallinie des CIE inklusive Purpur-Linie

## Hinweise

- Kann nur innerhalb eines React-Three-Fiber Canvas verwendet werden

## Optimierungen

-

---

## GamutBounds

```
export const GamutBounds = (
  {
    showRec601,
    showRec709,
    showRec2020
  }) => {...}
```

**showRec601:** Farbraum-Grenzen für Rec.601 einblenden (true/false)

**showRec709:** Farbraum-Grenzen für Rec.709 einblenden (true/false)

**showRec2020:** Farbraum-Grenzen für Rec.2020 einblenden (true/false)

- Zeichnet Farbraum-Grenzen in die CIE-Normfarbtafel

#### Hinweise

- Kann nur innerhalb eines React-Three-Fiber Canvas verwendet werden

#### Optimierungen

--

---

## GamutLabels

```
export const GamutLabels = (
  {
    showRec601,
    showRec709,
    showRec2020
  }) => {...}
```

**showRec601:** Farbraum-Beschriftung für Rec.601 einblenden (true/false)

**showRec709:** Farbraum-Beschriftung für Rec.709 einblenden (true/false)

**showRec2020:** Farbraum-Beschriftung für Rec.2020 einblenden (true/false)

- Zeigt eine Legende mit den eingeblendeten Farbraum-Grenzen am Bildschirmrand an

#### Hinweise

--

#### Optimierungen

- könnte etwas schöner aussehen, ggf. auch direkt an die Gamut-Grenzen schreiben
-

# Vektorskop

---

## VectorscopeView

```
export const VectorscopeView = (
  {
    signalYCBCR,
    withOverlays = false,
    encodedVideoStandard = 1,
    encodedBitDepthIdx = 0
  }) => {...}
```

**signalYCBCR:** siehe [Signal Arrays](#)

**withOverlays:** Buttons und Labels anzeigen (true/false)

**encodedVideoStandard:** Videostandard, dem signalYCBCR entspricht ([0..2] für Rec.601, 709 oder 2020)

**encodedBitDepthIdx:** Quantisierungsgrad, in dem signalYCBCR vorliegt ([0..2] für 8-, 10- oder 12-Bit)

- Rechnet Signal für Visualisierung um
- Stellt Einstellungs-Menüs für Visualisierung bereit

Verwendete GeneralComponents:

[ScopesCamera](#),  
[VideoStandardAlertView](#),  
[SettingsPopOverContainer](#)

---

## Subkomponenten

---

### VectorscopePlot

```
const VectorscopePlot = (
  {
    signalSmallYCBCR,
    signalRGB,
    useDiscreteSignalRepresentation
  }) => {...}
```

**signalSmallYCBCR, signalRGB:** siehe [Signal Arrays](#)

**useDiscreteSignalRepresentation:** Signal als diskrete Punkte oder als Linienzug darstellen (true/false)

- Bildet Bildpunkte des signalSmallYCBCR mit den entsprechenden Farben aus dem signalRGB im Vektorschop ab.

## Hinweise

- Kann nur innerhalb eines React-Three-Fiber Canvas verwendet werden

## Optimierungen

- Hier lässt sich die Performance noch deutlich optimieren, indem systematischer mit den Hooks "useMemo()", "useRef()",... gearbeitet wird und unter Verwendung eines [InstancedMesh](#). Hier muss aber auf die Paketversion von Three geachtet werden, da neuere Versionen mit anderen Bibliotheken (wie react-three-fiber) Probleme bereiten kann.

---

## LinePlot

```
export const LinePlot = ({ signalSmallYCBCR }) => {...}
```

**signalSmallYCBCR:** siehe [Signal Arrays](#)

- Zeichnet die Bildpunkte wie ein analoges Vektorschop als Linienzug über die Bildzeilen ein.

## Hinweise

- Kann nur innerhalb eines React-Three-Fiber Canvas verwendet werden

## Optimierungen

- Achsenbeschriftungen fehlen
- 

## VectorscopeBounds

```
export const CieBounVectorscopeBoundsds = () => {
```

**props:** --

- Zeichnet Achsen und Kreis mit Radius 1.

### Hinweise

- Kann nur innerhalb eines React-Three-Fiber Canvas verwendet werden

### Optimierungen

– Lässt sich mit den Anmerkungen zu [VectorscopePlot](#) ggf. noch optimieren.

---

## PeakSignalHexagon

```
export const PeakSignalHexagon = ({ videoStandard }) => {...}
```

**videoStandard:** Videostandard abgekürzt als String ("601", "709", "2020")

- Zeichnet abhängig vom Videostandard die Chroma-Signal-Grenzen als Hexagon ein.

### Hinweise

- Kann nur innerhalb eines React-Three-Fiber Canvas verwendet werden

### Optimierungen

– Lässt sich mit den Anmerkungen zu [VectorscopePlot](#) ggf. noch optimieren.

---

# Waveformmonitor

---

## WfmView

```
export const WfmView = (
  {
    signalYCBCR,
    withOverlays = false,
    encodedVidStdIdx = 1,
    encodedBitDepthIdx = 0
  }) => {...}
```

**signalYCBCR:** siehe [Signal Arrays](#)

**withOverlays:** Buttons und Labels anzeigen (true/false)

**encodedVidStdIdx:** Index des Videostandards, dem signalYCBCR entspricht ([0..2] für Rec.601, 709 oder 2020)

**encodedBitDepthIdx:** Quantisierungsgrad, in dem signalYCBCR vorliegt ([0..2] für 8-, 10- oder 12-Bit)

- Rechnet Signal für Visualisierung um
- Stellt Einstellungs-Menüs für Visualisierung bereit

Verwendete GeneralComponents:

[ScopesCamera](#),  
[VideoStandardAlertView](#),  
[SettingsPopOverContainer](#)

---

## Subkomponenten

---

## WfmPlot

```
const WfmPlot = (
  {
    signalSmallYCBCR,
    signalRGB,
    representationID,
    aspectRatio = 1.78
  }) => {...}
```

**signalSmallYCBCR, signalRGB:** siehe [Signal Arrays](#)

**representationID:** Wechsel zwischen RGB-, YCBCR-, Luma-Darstellung (0, 1, 2)

**aspectRatio:** Seitenverhältnis des Plots

- Bildet Bildpunkte des signalSmallYCBCR mit den entsprechenden Farben aus dem signalRGB als Waveformdarstellung ab. Dabei werden die Bildpunkte horizontal auf die Breite eines Vollbilds gestreckt.

Hinweise

- Kann nur innerhalb eines React-Three-Fiber Canvas verwendet werden

Optimierungen

- Hier lässt sich die Performance noch deutlich optimieren, indem systematischer mit den Hooks "useMemo()", "useRef()",... gearbeitet wird und unter Verwendung eines [InstancedMesh](#). Hier muss aber auf die Paketversion von Three geachtet werden, da neuere Versionen mit anderen Bibliotheken (wie react-three-fiber) Probleme bereiten kann.

---

signalToWfmArray

```
function signalToWfmArray(
  signalArray,
  channelIdx,
  hexColorString = "#555",
  paradePosition = undefined,
  withChromaOffset = false,
  aspectRatio = 1.78
){...}
```

**signalArray:** signalSmallYCBCR oder signalRGB, siehe [Signal Arrays](#)

**channelIdx:** Auswahl des Signal-Kanals (0,1,2)

**hexColorString:** Farbe, mit der der Kanal im WfmPlot abgebildet werden soll  
(hexadezimal-String)

**paradePosition:** Position in der Paraden-Darstellung. Bei undefined wird es über die volle WFM-Breite abgebildet (undefined, 0, 1, 2)

**withChromaOffset:** Vertikaler Versatz der Nulllinie auf 50% für Chroma-Komponenten  
(true/false)

**aspectRatio:** Bildseitenverhältnis der Waveform-Darstellung, in der es abgebildet wird

- Erstellt aus einem Signal-Kanal ein Array, mit dessen Koordinaten für eine Waveform-Darstellung.
- Zu drei Raum-Koordinaten kommt jeweils noch die Farbe, in der der Kanal im WfmPlot dargestellt werden soll.

## Hinweise

- Die z-Koordinate wird nur zur besseren Weiterverarbeitung im WfmPlot hinzugenommen.
- Könnte auch als Arrow-Function geschrieben werden. Zur Unterscheidung, dass es nur eine Berechnung ausführt wird es als "Reguläre" Funktion geschrieben.

## Optimierungen

--

---

## WfmGrid

```
export const WfmGrid = ({ aspectRatio = 1.78 }) => {
```

**aspectRatio:** Bildseitenverhältnis der Waveform-Darstellung, in der es abgebildet wird

- Zeichnet horizontale Linien für eine 10%-Skallierung.

## Hinweise

- Kann nur innerhalb eines React-Three-Fiber Canvas verwendet werden

## Optimierungen

- Lässt sich mit den Anmerkungen zu [WfmPlot](#) ggf. noch optimieren.
-

# Signal-Generator

---

## SignalGeneratorView

```
export const SignalGeneratorView = (
  {
    setSignal,
    setEncodingVideoStandard,
    setEncodingBitDepthIdx,
    showHideButton = false,
    style=undefined
  }) => {...}
```

**setSignal:** Signal-Ausgang: Übergibt ein signalYCBCR siehe [Signal Arrays](#) über diesen Parameter an Eltern-Objekt.

**setEncodingVideoStandard:** Index des Videostandards, dem signalYCBCR entspricht ([0..2] für Rec.601, 709 oder 2020)

**setEncodingBitDepthIdx:** Index des Quantisierungsgrads, dem signalYCBCR entspricht ([0..2] für 8-, 10- oder 12-Bit)

**showHideButton:** Button zum Ausblenden des Generators einblenden (true/false)

**style:** Zum annehmen von Style-Parametern

- Errechnet ein signalYCBCR aus einem signalSmallRGB.
- Buttons zum Wechsel des Videostandards und Quantisierung des signalYCBCR
- Beinhaltet einen GeneratorContainer um das signalSmallRGB zu beschreiben.
- Beinhaltet Corrector um das signalSmallRGB zu verändern.

## Optimierungen

- Benutzeroberfläche überarbeiten und Platz besser ausnutzen, damit die ScrollView weg kann.
- Wenn ScrollView weg ist und Performance optimiert wurde können Slider zum Einstellen der entsprechenden Parameter verwendet werden.
- Zur Vereinheitlichung den Parameter "setSignal" in "setSignalYCBCR" und "setEncodingVideoStandard" in "setEncodingVidStdIdx" umbenennen (und entsprechende Parameter in den Screens). Aus unerklärlichen Gründen führt dies zu dem Fehler "setSignalYCBCR is not a function.".

Verwendete GeneralComponents:

--

---

## Subkomponenten

---

### GeneratorContainer

```
const GeneratorContainer = (
  {
    setSignalSmallRGB,
    generatorIdx,
    setGeneratorIdx
  }) => {...}
```

**setSignalSmallRGB:** Signal-Ausgang: Übergibt ein signalSmallRGB (siehe [Signal Arrays](#)) über diesen Parameter an Eltern-Objekt.

**generatorIdx:** Index des Generators, der verwendet und angezeigt werden soll.

**setGeneratorIdx:** Veränderung des Generator-Index beim Eltern-Objekt.

- Sammelt alle Generator-Typen und stellt Schaltflächen zum Wechsel des anzuzeigenden Generators bereit.

### Hinweise

--

### Optimierungen

- Ggf. mittels "[Containment](#)" umsetzen, um alles kompakter und flexibler für neue Signalarten zu machen. Gleiches Prinzip wie mit den Settings-Objekten.
-

## FullColorGenerator

```
export const FullColorGenerator = ({ setSignalSmallRGB })=>
{...}
```

**setSignalSmallRGB:** Signal-Ausgang: Übergibt ein signalSmallRGB (siehe [Signal Arrays](#)) über diesen Parameter an Eltern-Objekt.

- Erzeugt ein einfarbiges Vollbild als signalSmallRGB.
- Benutzeroberfläche mit RGB- und HSV-Modus.

### Hinweise

- showingRgbControls dient neben dem Wechsel der Anzeige vor Allem dazu, dass sich die useLayoutEffect()-Hooks nicht in einer Endlosschleife verfangen.

### Optimierungen

- TapButtons durch Slider ersetzen wenn Optimierungen in [SignalGeneratorView](#) erfüllt sind.
- Option um Überpegel mit den RGB-Parametern zu erzeugen.

---

## GradientGenerator

```
export const GradientGenerator = ({ setSignalSmallRGB })=>
{...}
```

**setSignalSmallRGB:** Signal-Ausgang: Übergibt ein signalSmallRGB siehe [Signal Arrays](#) über diesen Parameter an Eltern-Objekt.

- Erzeugt ein Farbverlauf als signalSmallRGB.
- Wechsel von horizontalen und vertikalen Verlauf
- Schnellauswahl wichtiger Farbwerte durch Button-Matrix (ColorPad).

### Hinweise

--

## Optimierungen

- Ggf. zusätzliche Ansicht mit detaillierteren Einstellungsmöglichkeiten der Farbwerte (mit RGB und HSV-Slidern)
- 

## BarsGenerator

```
export const BarsGenerator = ({ setSignalSmallRGB })=> {....}
```

**setSignalSmallRGB:** Signal-Ausgang: Übergibt ein signalSmallRGB siehe [Signal Arrays](#) über diesen Parameter an Eltern-Objekt.

- Erzeugt EBU-Farbbalken als signalSmallRGB.
- Benutzeroberfläche mit Wechsel zwischen 100/100 und 100/75

## Hinweise

--

## Optimierungen

- Weitere Test-Bilder
- 

## Corrector

```
export const Corrector = (
  {
    contrastOffset,
    setContrastOffset,
    gammaOffset,
    setGammaOffset,
    brightnessOffset,
    setBrightnessOffset
  }) => {
```

**contrastOffset:** Aktueller Wert des Kontrast-Versatz-Parameters des Eltern-Objekts.

**setContrastOffset:** Übergibt neuen Wert des Kontrast-Versatz-Parameters ans Eltern-Objekt.

**gammaOffset:** Aktueller Wert des Gamma-Versatz-Parameters des Eltern-Objekts.

**setGammaOffset:** Übergibt neuen Wert des Gamma-Versatz-Parameters ans Eltern-Objekt.

**brightnessOffset:** Aktueller Wert des Helligkeit-Versatz-Parameters des Eltern-Objekts.

**setBrightnessOffset:** Übergibt neuen Wert des Helligkeit-Versatz-Parameters ans Eltern-Objekt.

- Benutzeroberfläche zum Verändern der Corrector-Parameter im Eltern-Objekt.

## Hinweise

--

## Optimierungen

- TapButtons durch Slider ersetzen wenn Optimierungen in [SignalGeneratorView](#) erfüllt sind.
  - Sättigungs-Regler hinzufügen.
-

# Signal-Vorschau

---

## SignalPreviewView

```
export const SignalPreviewView = (
  {
    signalYCBCR,
    withOverlays = false,
    encodedVideoStandard = 1,
    encodedBitDepthIdx = 0
  }) => {...}
```

**signalYCBCR:** siehe [Signal Arrays](#)

**withOverlays:** Buttons und Labels anzeigen (true/false)

**encodedVideoStandard:** Videostandard, dem signalYCBCR entspricht ([0..2] für Rec.601, 709 oder 2020)

**encodedBitDepthIdx:** Quantisierungsgrad, in dem signalYCBCR vorliegt ([0..2] für 8-, 10- oder 12-Bit)

- Rechnet Signal für Visualisierung um
- Stellt Einstellungs-Menüs für Visualisierung bereit

### Optimierungen

- Settings und VideoStandardAlertView einbinden
- Hinzufügen von Videostandard-Wechsel erfordert, dass für die Vorschau ein Wechsel zwischen relativer Farb-Darstellung und absoluter Farb-Darstellung besteht (alles über sRGB hinaus klappt)

Verwendete GeneralComponents:

--

---

## Subkomponenten

---

## SignalPreviewPlot

```
const SignalPreviewPlot = (
  {
    signalSmallRGB,
    signalRGB = undefined,
    signalYCBCR = undefined
    labelIdx = 0
  }) => {...}
```

**signalSmallRGB, signalRGB, signalYCBCR:** siehe [Signal Arrays](#)

**labelIdx:** Auswahl welche Beschriftung auf Bildpunkt-Gruppe eingeblendet werden soll:  
keins, signalRGB oder signalYCBCR (0, 1, 2)

- Erzeugt eine Vorschau des signalSmallRGB, indem die Bildpunkt-Werte auf ein Vollbild gestreckt werden.
- Blendet optional die Signalwerte von signalRGB oder signalYCBCR ein, welche sich an den Videostandards der Rec.601, 709 und 2020 orientieren.

### Hinweise

- Kann außerhalb von React-Three-Fiber Canvas verwendet werden, weil nnur React Native Views verwendet werden.

### Optimierungen

- Hier lässt sich die Performance wahrscheinlich noch optimieren, indem systematischer mit den Hooks "useMemo()", "useRef()",... gearbeitet wird.
- Manchmal erscheinen weiße Linien zwischen den Bildpunkt-Gruppen.

---

# Einstellungs-Fenster

---

## SettingsPopOverContainer

```
export const SettingsPopOverContainer = (props) => {...}
```

**props.children:** Einstellungs-Elemente

**props.setSettingsVisible:** Übergibt dem Eltern-Objekt mit "false" den Befehl zum Schließen.

- Einstellungs-Fenster, das als Container für Einstellungs-Elemente fungiert.

### Hinweise

- (props.children) ist ein von React reservierter Parameter um React-Elemente übergeben zu bekommen. Siehe [Containment - React Dokumentation](#)

Verwendete GeneralComponents:

--

---

# Subkomponenten

---

## ToggleElement

```
const ToggleElement = (
  {
    elementTitle,
    title,
    onPress,
  }) => {...}
```

**elementTitle:** Titel des Elements

**title:** Titel des Buttons

**onPress:** Funktion, die im Eltern-Objekt des SettingsPopOverContainers beim Anklicken ausgeführt wird.

- Allgemein nutzbarer, einfacher Button.

## Hinweise

- braucht einen SettingsPopOverContainer.

## Optimierungen

--

---

## GamutSelectElement

```
export const GamutSelectElement = (
  {
    showRec601,
    toggleRec601,
    showRec709,
    toggleRec709,
    showRec2020,
    toggleRec2020
  }) => {...}
```

**showRec601:** Sichtbarkeit der Rec.601-Gamuts-Grenzen im Elternobjekt des SettingsPopOverContainers (true/false)

**toggleRec601:** Funktion zum Wechsel der Sichtbarkeit der Rec.601-Gamuts-Grenzen

**showRec709:** Sichtbarkeit der Rec.709-Gamuts-Grenzen im Elternobjekt des SettingsPopOverContainers (true/false)

**toggleRec709:** Funktion zum Wechsel der Sichtbarkeit der Rec.709-Gamuts-Grenzen

**showRec2020:** Sichtbarkeit der Rec.2020-Gamuts-Grenzen im Elternobjekt des SettingsPopOverContainers (true/false)

**toggleRec2020:** Funktion zum Wechsel der Sichtbarkeit der Rec.2020-Gamuts-Grenzen

- Element um die Gamut-Grenzen für die CIE-Normfarbtafel auszuwählen.

## Hinweise

- braucht einen SettingsPopOverContainer.

## Optimierungen

--

---

## VideoStandardSelectElement

```
export const VideoStandardSelectElement = (
  {
    vidStdIdx,
    setVidStdIdx,
    bitDepthIdx,
    setBitDepthIdx
  }) => {
```

**aspectRatio:** Bildseitenverhältnis der Waveform-Darstellung, in der es abgebildet wird

- Zum Wechseln des Videostandards und des Quantisierungsgrades im Eltern-Objekt des SettingsPopOverContainers.

## Hinweise

- braucht einen SettingsPopOverContainer.

## Optimierungen

--

---

# Allgemeine Komponenten (Kurze)

---

## ScopesCamera

```
export const ScopesCamera = (
  {
    position,
    target = [0, 0, 0],
    initialZoomScale = 1,
    zoomOffset = 0
  }) => {...}
```

**position:** 3D Position der Kamera (Array: [ x, y, z])

**target:** Punkt auf den die Kamera Blickt (Array: [ x, y, z])

**initialZoomScale:** Zum anpassen des Standard-Zooms der Kamera, je nach Visualisierung (Fließkommazahl)

**zoomOffset:** Relativer Versatz des Kamera-Zooms (Fließkommazahl)

- Eine React-Three-Fiber-Kamera, mit angepassten Funktionen für die Abbildung von Messansichten
- Orthographische projizierend, um Messvisualisierungen nicht zu verzerren.

## Hinweise

- Kann nur innerhalb eines React-Three-Fiber Canvas verwendet werden

## Optimierungen

- Für die Nutzung in 3D CIE-Normfarbtafel ggf. Wechsel zur perspektivischen Projektion zur besseren Orientierung.

---

## VideoStandardAlertView

```
export const VideoStandardAlertView = (
  {
    signalVidStdIdx,
```

```
scopeVidStdIdx,  
signalBitDepthIdx = 0,  
scopeBitDepthIdx = 0  
) => {...}
```

**signalVidStdIdx:** Videostandard des erhaltenen Videosignals (Rec.601, 709, 2020 als Index 0...2)

**scopeVidStdIdx:** Videostandard, die in der Messansicht eingestellt ist (Rec.601, 709, 2020 als Index 0...2)

**signalBitDepthIdx:** Quantisierungsgrad, des erhaltenen Videosignals ([0..2] für 8-, 10- oder 12-Bit)

**scopeBitDepthIdx:** Quantisierungsgrad, die in der Messansicht eingestellt ist ([0..2] für 8-, 10- oder 12-Bit)

- Blendet einen Hinweis ein wenn die Messansicht einen anderen Videostandard bzw. Quantisierungsstufe erwartet als für das Signal verwendet wurde.

## Hinweise

--

## Optimierungen

--

---

# App-Navigation

---

## Navigation

```
export default function Navigation() {...}
```

- Erstellt einen DrawerNavigator
- Jeder Screen ist in einem StackNavigator verpackt, da dies die Header-Bar bereitstellt. So kann man zentral vom Navigator aus alle Simulations-Header-Bars beschriften und auf Knopdruck den den DrawerNavigator einblenden.

## Optimierungen

- Einen Weg finden um die StackWrap-Funktionen kompakter mit weniger Code-Wiederholungen zu erzeugen.
- Eigene Icons erstellen, die besser zu den Simulationen passen.
- Eine Untergruppierung von Simulationen zu einzelnen Themengebieten. Bisher könnte man alle als "Signal & Messung" Betiteln.

# Screens

---

## CIE-Normfarbtafel

```
export default function CieScreen() {
```

Farbmodell zur Visualisierung aller vom menschlichen Auge wahrnehmbaren Farben (in 3D Ansicht) bzw. Farbwerte (in xy-Ansicht).

### Verwendete Komponenten

- [CIEView](#)
- [SignalPreviewView](#)
- [SignalGeneratorView](#)

### Ideen für neue Funktionen

- "Orbit-Controls" für die Bewegung der Kameraperspektive in 3D
- 3D Gamut-Körper einblenden
- Einstellungen für die Punkt-Farbe. Also ob Farben relativ zum Farbraum dargestellt werden sollen oder ob sie immer nur den in sRGB abbildbaren Bereich in Farbe darstellen soll.

---

## Waveformmonitor

```
export default function WfmScreen() {
```

Instrument zur Betrachtung der Pegel eines Videosignals. Beinhaltet Darstellungsformen als Komponentensignal, Luma-Signal und RGB. Darstellungen als Paraden

### Verwendete Komponenten

- [WFMView](#)
- [SignalPreviewView](#)

- [SignalGeneratorView](#)

## Ideen für neue Funktionen

- Lineinzug-Darstellung, wie im Vektorskop
  - Overlay-Ansichten
- 

## Vektorskop

```
export default function VectorScopeScreen() {
```

Instrument zur Betrachtung des Chrominanzanteils eines Komponentensignals.

### Verwendete Komponenten

- [VectorScopeView](#)
- [SignalPreviewView](#)
- [SignalGeneratorView](#)

## Ideen für neue Funktionen

- Option zur Anzeige von Toleranzfeldern
- 

## Komponentensignal

```
export default function SignalPreviewScreen() {
```

Eine Vollbildansicht der Signalvorschau zur genaueren Betrachtung der erzeugten Signale.  
Beinhaltet eine Beschriftung über die Signalwerte in RGB und als Komponentensignal.

### Verwendete Komponenten

- [SignalPreviewView](#)

- [SignalGeneratorView](#)

## Ideen für neue Funktionen

- Bessere Formatierung der Beschriftungen. Speziell bei horizontalen Farbverläufen im Signal
- 

## Scopes Übersicht

```
export default function ScopesCombinationScreen() {
```

Gesamtübersicht über alle Messinstrumente. Zeigt in welcher Beziehung diese zueinander stehen.

## Verwendete Komponenten

- [CIEView](#)
- [WFMView](#)
- [VectorscopeView](#)
- [SignalPreviewView](#)
- [SignalGeneratorView](#)

## Ideen für neue Funktionen

- Die vertikalen Zwischenräume der Messansichten können mehr gestaucht werden, um dem Signalgenerator mehr Platz zu geben.
- Ggf. ein allgemeines Einstellungsmenü zum Wechsel aller Videostandard

--

# Signal-Arrays

---

Zur Kommunikation zwischen den Komponenten und bei internen Berechnungen gibt es wiederkehrende Signalarten, die mit deren Notation hier zusammengefasst sind.

Allgemein gilt:

- Aus Gründen der Datenersparnis können Höhe und Breite des Arrays von der Auflösung der Videostandards abweichen. Sie repräsentieren dennoch ein Vollbild und sind als relative Bildpositionen zu betrachten. Für eine korrekte Darstellung muss das Array auf ein Vollbild gestreckt werden (meist 16:9).

## Allgemeiner Signal-Aufbau:

signalXXX [rowIdx] [columnIdx] [subPixelIdx]

rowIdx	Index der Zeile. Entspricht relativer vertikaler Bildposition.	[0...]
columnIdx	Index der Spalte. Entspricht relativer horizontaler Bildposition.	[0...]
subPixelIdx	Index der XXX-Subpixel eines Bildpunkts bzw. Bildfläche.	[0...2]

Beispiel:

```
[ [ [X X X],   [X X X],   [X X X] ],
  [ [X X X],   [X X X],   [X X X] ],
  [ [X X X],   [X X X],   [X X X] ] ]
```

## signalYCBCR

- Haupt-Kommunikations-Array zwischen App-Komponenten der Signalerzeugung oder -Messung.
- Repräsentiert Komponentensignal in 8-, 10- oder 12-Bit nach Rec.601, Rec.709 oder Rec.2020.
- Der verwendete Videostandard und Quantisierungsgrad müssen separat mitgeteilt werden

- Subpixel: Y, Cr, Cb

<b>Subpixel-Werte</b>	<b>8-Bit</b>	<b>10-Bit</b>	<b>12-Bit</b>
Y Peak	235	940	3760
Y Schwarz	16	64	256
CrCb Farblos	128	512	2048
CrCb Max Farbe	16 & 240	64 & 960	256 & 3840

(Siehe ITU-R BT.601, ITU-R BT.709 und ITU-R BT.2020)

Beispiel 8-Bit Quantisierung (Schwarz-Weiß-Muster):

```
[ [ [16 128 128], [235 128 128], [16 128 128], [235 128
128] ],
  [ [235 128 128], [16 128 128], [235 128 128], [16 128
128] ],
  [ [16 128 128], [235 128 128], [16 128 128], [235 128
128] ] ]
```

## signalSmallYCBCR

- Wie *signal/YCBCR*, aber Subpixel sind unabhängig von Quantisierungs-Grad normiert (legaler Videobereich).
- Entsteht als Zwischenschritt der Signalumrechnung:

signalSmallIRGB -> signalSmallYCBCR -> signalYCBCR

- Findet zur Darstellung in Messinstrumenten Anwendung

Y Peak	1
Y Schwarz	0
CrCb Farblos	0
CrCb Voll*	C < 0 & C > 0

\* Genaue Grenzen hängen von Videostandard ab

## signalRGB

- Repräsentiert RGB-Signal in 8-, 10- oder 12-Bit nach Rec.601, Rec.709 oder Rec.2020.
- Der verwendete Videostandard und Quantisierungsgrad müssen separat mitgeteilt werden
- Subpixel: R, G, B

Subpixel-Werte	8-Bit	10-Bit	12-Bit
R,G,B Peak	235	940	3760
R,G,B Schwarz	16	64	256

(Siehe ITU-R BT.601, ITU-R BT.709 und ITU-R BT.2020)

## signalSmallRGB

- Wie *signalRGB*, aber Subpixel sind unabhängig von Quantisierungs-Grad normiert (legaler Videobereich).
- Subpixel: R, G, B

R,G,B Peak	1
R,G,B Schwarz	0

Beispiel Schwarz-Weiß-Muster:

```
[ [ [0 0 0], [1 1 1] ],
  [ [1 1 1], [0 0 0] ],
  [ [0 0 0], [1 1 1] ] ]
```

## signalXYZ

- Signal, in dem einzelne Bildpunkte als Farbwerte in XYZ-System notiert sind

## signalxyY

- Signal, in dem einzelne Bildpunkte als Farbwerte in xyY-System notiert sind.
- zur Visualisierung in CIE-Normfarbtafel.