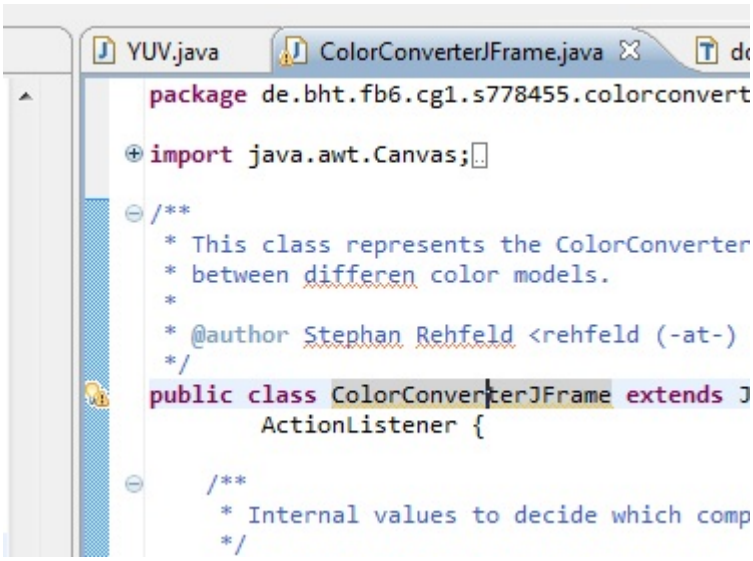

CG Übung 1: ColorConverter

Sascha Feldmann
sascha.feldmann@gmx.de

Inhaltsverzeichnis

1 Profil

1.1 Aufgabe

Erstellung eines ColorConverters.

1.2 Abgabe

24.04.2012

1.3 Autor

Sascha Feldmann - 778455

1.4 Beginn der Bearbeitung

05.04.2012

2 Dokumentation zur Aufgabe

2.1 Methodik

Zuerst wurden die Farbräume von mir modelliert. Diese sollten alle folgendermaßen linear aufgebaut sein:

- Konstante Attribute für die Farbkomponenten

- Kapselung der Farbkomponenten

- Kontrolle über den Wertebereich und keine Möglichkeit der nachträglichen Änderung von Farbkomponenten Methoden, die den aktuellen Farbraum in einen anderen konvertieren und die Instanz zurückgeben

Anschließend wurde die GUI um den Farbraum CMY erweitert. Dazu musste die vorhandene Ereignismethode um die Funktionalität der weiteren Slider erweitert werden.

Nach der Anpassung der GUI musste die Konvertierung in die anderen Farbräume gesteuert werden. Mit dem Bewegen eines Sliders sollten alle anderen Farbräume entsprechend angepasst werden. Dieses Hauptproblem wird nachfolgend erläutert.

2.1.1 Konvertierung der Farbräume

Besonders leicht zu realisieren ist die Konvertierung von CMY-Komponenten in RGB-Komponenten und umgekehrt.

Schwieriger war die Konvertierung zwischen YUV und RGB-Farbräumen.

Daher bietet der RGB-Farbraum zentral alle Transformationsmethoden (RGB-YUV und RGB-CMY), YUV sowie CMY konvertieren nur zu RGB. Ändert sich also z.B. die YUV-Komponente, so wird erst die RGB-Komponente aktualisiert, welche dann die CMY-Komponente konvertiert.

Die Konvertierung von YUV zu RGB erfolgt durch eine Matrixrechnung:

$$M = \begin{pmatrix} 1.0 & 0.0 & 1.140 \\ 1.0 & -0.394 & -0.581 \\ 1.0 & 2.028 & 0.0 \end{pmatrix} \cdot \begin{pmatrix} Y \\ U \\ V \end{pmatrix}$$

Dabei entstehen unausweichlich float-Werte, die Java-Math-Funktionen auf integer-Werte gerundet werden müssen, mit welchen das Programm arbeitet.

Auch die reverse Umrechnung von RGB- in YUV-Werten bedarf einer Transformationsmatrix. Diese erhält man durch Inversion der YUV-¿RGB-Matrix. Die Farbanteile werden zunächst in die Helligkeitskomponente Y, die Cyan-Rot-Balance U und die Gelb-Blau-Balance V umgewandelt.

$$M = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.437 \\ 0.615 & -0.515 & -0.100 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

2.2 Implementierung der Farbräume

Nachfolgend wird die konkrete Implementierung der Farbraum-Klassen erläutert. Dabei gehe ich insbesondere auf die Konvertierung ein.

2.2.1 RGB

Listing 1: returnCMY-Methode

```
1  /**
2           * Convert the RGB values to CMY.
3           * @return an CMY object
4           */
5          public CMY returnCMY() {
6                  int c = MAX - red;
7                  int m = MAX - green;
8                  int y = MAX - blue;
9                  CMY cmy = new CMY(c, m, y );
10                 return cmy;
11         }
```

Die Konvertierung vom additiven Farbmodell RGB zum subtraktiven CMY erfolgt einfach durch Subtraktion der RGB-Komponenten vom maximalen Wert (i.d.R. ist MAX = 255, Standard-Bittiefe bei TrueColor, da $2^8 = 256$). Diese einfache Umrechnung ist auf die Nutzung von Farben im CMY-Farbraum zurüruckzuführen. Hier wird Licht abgezogen/subtrahiert/absorbiert. Cyan absorbiert Rot am stärksten, der Gesamtmischung wird also der entsprechende Rot-Absorptions-Anteil abgezogen. Durch Mischung von Subtraktionsanteilen entstehen dann konträr zu RGB unterschiedliche Farbmischungen.

Betrachtet man das YUV- und RGB-Farbmodell als Würfel, so stellt man entsprechend der o.g. Fakten fest, dass die jeweils gegenüberliegenden Farben sich wegsubtrahieren.

Für die Darstellung von Weißbedeutet das Folgendes:

- Im RGB-Farbmodell wird Weißals maximale Reflektion aller Farbkomponenten interpretiert.

- Im YUV-Farbmodell ist Weißdie minimale Absorption aller Farbkomponenten. Es wird also kein Farbanteil absorbiert.

Listing 2: returnYUV-Methode

```
12 /**
13          * Convert the RGB values into a new YUV
14          instance
15          * @return the YUV instance
16          */
17         public YUV returnYUV() {
18                 final float yf = 0.299f*red
19                 + 0.587f * green + 0.114f * blue;
20                 final float uf = (blue - yf)*0.493f;
21                 final float vf = (red - yf)*0.877f;
22
23                 final int y = Math.round(yf);
24                 final int u = Math.round(uf);
25                 final int v = Math.round(vf);
26
27                 final YUV yuv = new YUV(y, u, v );
28
29                 return yuv;
30         }
```

Die Konvertierung vom RGB- zum YUV-Modell setzt nicht die Matrixrechnung aus Absatz 1 um. Die angewandte Methode ist einfacher zu im-

plementieren und funktioniert nach diesem Prinzip:

- Die Luminanzkomponente Y ist die Summe der Gewichtungen der Rot-, Grün und Blau-Anteile.

- Die Chromakomponenten U und V sind die Gewichtung der Differenz zwischen Blau- bzw. Rot-Wert und der Luminanzkomponente Y.

Das YUV-Modell macht sich also das optische Phänomen zunutze, dass der Grün-Anteil heller wahrgenommen wird als Rot- oder Blau-Anteile.

2.2.2 YUV

Die YUV-Farbraum-Klasse bietet nur eine Methode zur Konvertierung von YUV zu RGB an. Dabei wird die inverse Transformationsmatrix (s. Method-Abschnitt) angewendet.

Listing 3: returnRGB-Methode

```
31 /**
32          * Convert values into a new RGB instance
33          * and return it.
34          * @return an RGB object with the
35                converted values.
36          */
37         public RGB returnRGB() {
38                final float bf = y + u / 0.493f;
39                final float rf = y + v/0.877f;
40                final float gf = y - 0.39466f * u
41                                          - 0.5806f * v;
42
43                final int r = Math.round(rf );
44                final int g = Math.round(gf);
45                final int b = Math.round(bf);
46
47                final RGB rgb = new RGB(r, g, b);
48                return rgb;
49         }
```

2.2.3 Farbraum-Kapselung

Jede Farbraum-Klasse sorgt dafür, dass die mögliche Farbgrenzen eingehalten werden. Da jeder Farbraum final erzeugt wird, erfolgt dies im Konstruktor.

Listing 4: Konstruktor der YUV-Klasse

```
50 /**
51          * Create a new instance of the
52          colorspace with uneditable values.
53          * @param y the luminance - a
54          value between 0 and  255
55          * @param u cyan-red-balance - a
56          value between -111 and 111
57          * @param v yellow-blue-balance - a
58           value between -157 and 157
59          *  @throws IllegalArgumentException
60          if the values are outside the limits.
61          */
62         public YUV(final int y, int u, int v) {
63                 if(y < 0 || y > YMAX)
64                 throw new IllegalArgumentException
65                 ("Invalid value for y in YUV, value"+
66                 "+y);
67 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣if(u␣<␣UMIN␣||␣u␣>␣UMAX␣)
68 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣throw␣new␣IllegalArgumentException
69 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣("Invalid value for u in YUV,"+
70 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" value "+u);
71 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣if(v␣<␣VMIN␣||␣v␣>␣VMAX␣)
72 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣throw␣new␣IllegalArgumentException
73 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣("Invalid value for v in YUV,"+
74 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" value "+v);
75 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣this.y␣=␣y;
76 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣this.u␣=␣u;
77 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣this.v␣=␣v;
78 ␣␣␣␣␣␣␣␣}
```

Im YUV-Farbraum ist die Luminanz wie bei RGB- und YUV bei einer Bittiefe von 8 Bit auf 255 begrenzt. Bei Weiß(RGB: 255 255 255) wird also die maximale Luminanz erreicht.

Der Chroma-Anteil U (Cyan-Rot-Gewichtung) kann -111 nicht unterschreiten bzw. 111 überschreiten. Dazu betrachten wir die Minima bzw. Maxima ausgehend vom Blau-Anteil des RGB-Farbmodells:

- Wenn kein Blau-Anteil vorhanden ist, ist U minimal, also $B = 0$. Für

U bedeutet das:
$U_{min} = (0 - Y_{max}) \cdot 0.493f$
Y$_{max}$ ist in diesem Beispiel, wenn R und G maximal und B minimal sind:
$Y_{max} = 0.299 \cdot 255 + 0.587 \cdot 255 = 226$
U$_{min}$ ist dann also:
$U_{min} = (0 - 226) \cdot 0.493f = -111$

- Wenn der Blau-Anteil maximal und R und G minimal sind, bedeutet dies für
  $Y_{max} = 0.114 \cdot 255 = 29$
  U$_{max}$ ist dann also:
  $U_{max} = (255 - 29) \cdot 0.493f = 111$

Die Klasse YUV muss also für die Integrität sorgen. Daher wird eine IllegalArgumentException geworfen, wenn aufgrund eines Fehlverhaltens z.B. logischer Natur die mathematisch festgelegten Grenzen über -oder unterschritten werden. Um Rechenzeit zu sparen, wurden die Minimas und Maximas als Konstanten festgelegt.

Für den Chroma-Anteil ändert sich an der Berechnung der Minima und Maxima die Werte, da Rot heller wahrgenommen wird.
$Y_{max} = 0.587 \cdot 255 + 0.114 \cdot 255 = 178$
$V_{min} = (0 - Y_{max}) \cdot 0.877f1$
$V_{min} = (0 - 178) \cdot 0.877 = -156$
Für die Maxima von V gilt: $Y_{min} = 0.299 \cdot 255 = 76$
$V_{max} = (255 - Y_{max}) \cdot 0.877f1$
$V_{max} = (255 - 76) \cdot 0.877 = 156$