

ERWEITERUNG: BIT-EBENEN



Bit-Ebenen

Einen weiteren interessanten Aspekt kann man untersuchen, wenn man die Intensitäten als 8-Bit-Binärzahlen auffasst. Man kann das (Graustufen-)Bild dann in acht verschiedene Bit-Ebenen aufteilen. Möchte man z.B. die vierte Bitebene ($2^3 = 8$) haben, setzt man alle anderen Bits auf 0. Dies erreicht man am leichtesten mit einer Und-Verknüpfung:

$$I_{neu} = I_{alt} \text{ UND}_{bitweise} 8$$

	Dezimal	Binär							
I_{alt}	190	1	0	1	1	1	1	0	1
Bitebene 4	8	0	0	0	0	1	0	0	0
I_{neu}	8	0	0	0	0	1	0	0	0

Die niederwertigen Bitebenen sehen auf den ersten Blick komplett schwarz aus, da die maximale Helligkeit nur 1 (1. Bit-Ebene), 2 (2. Bit-Ebene) oder 4 (3. Bit-Ebene) beträgt. Daher muss man die Bilder in Schwarz-Weiß-Bilder umwandeln, damit man etwas erkennt.

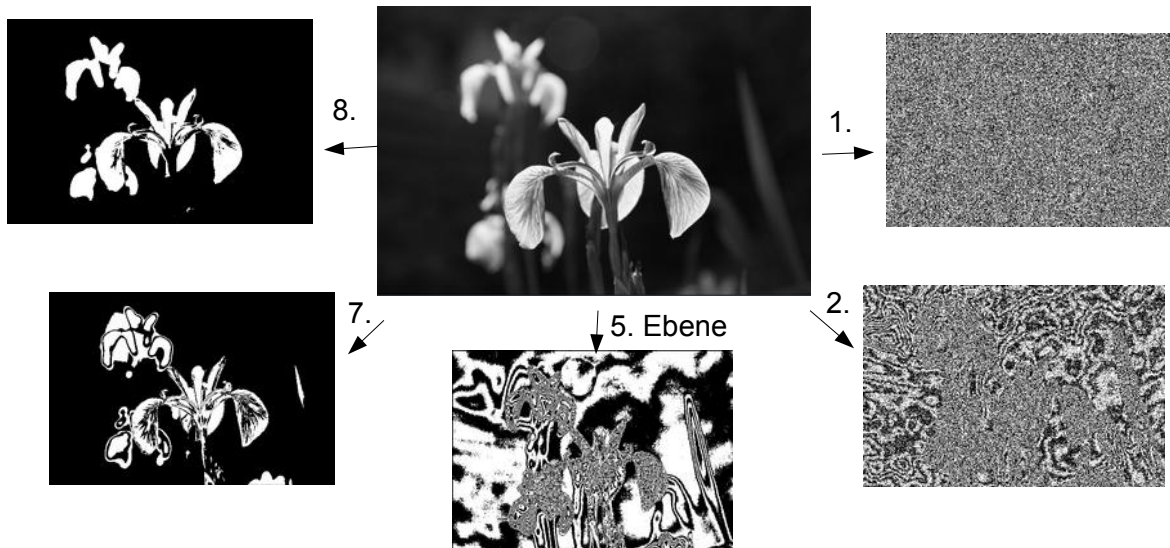
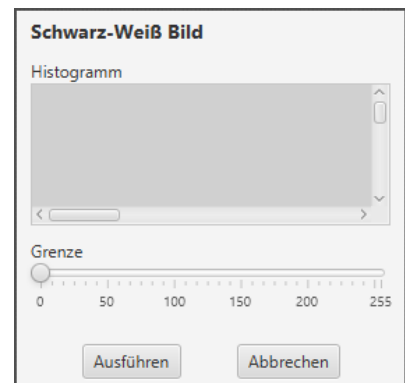


Abb. 1: Bitebenen eines Graustufenbildes (eigenes Werk)

Aufgaben

1. Erzeuge eine Klasse Bitebenen. Implementiere eine Methode `public Picture schwarzweiss(Picture originalbild, int grenze)`, die ein Bild in ein Schwarz-Weiß-Bild umwandelt. Alle Pixel mit einer Helligkeit unterhalb der Grenze, werden schwarz dargestellt. Alle anderen weiß. Ermittle die Helligkeit eines Pixel mit einer der Methoden, die bei der Umwandlung in Graustufen besprochen wurden.
2. Erstelle im Farben-Menü einen neuen Menüpunkt "Schwarz-Weiß-Bild". Rufe in der entsprechende Action-Methode die erstellte Methode mit einer festen Grenze auf. Alternativ kannst du auch ein Optionsfenster erstellen, in dem man die Grenze einstellen kann. Falls du die Erweiterung Histogramme bearbeitet hast, kann dieses Optionsfenster ein Histogramm enthalten. Andernfalls enthält es nur einen Slider für die Grenze. Wird dieser bewegt, wird die Darstellung des Bildes aktualisiert.





3. Implementiere eine Methode `public Picture bitEbene(Picture originalbild, int ebene)`, die die entsprechende Bitebene aus einem Bild extrahiert. Wende die oben beschriebene Formel auf jede Farbkomponente an.



Hinweis: In Java wird bitweises UND mit dem Operator `&` realisiert.

Hinweis: Die notwendige 2er-Potenz kann man berechnen, indem man eine Eins um (Ebene-1) Stellen in der Binärdarstellung nach links verschiebt. In Java: $(1 \ll (\text{ebene} - 1))$

4. Erstelle im Effekte-Menü einen neuen Menüpunkt "Bitebenen".



Erzeuge ein Optionsfenster, in dem man mit Checkboxes festlegen kann, welche Bitebenen angezeigt werden sollen. Eine weitere Checkbox soll festlegen, ob man am Ende das Bild in ein Schwarz-Weiß-Bild (Grenze 1) umwandeln möchte.

Beim Anklicken einer Checkbox soll die Anzeige aktualisiert werden. Lege dazu für die `selectedProperty` einen Action-Listener fest.

```
cbBitebene1.selectedProperty().addListener(  
    (observable, oldValue, newValue) ->  
        bitebenen(alt));
```

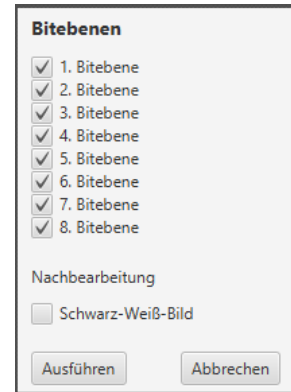
In der Methode `bitebene(Picture altesBild)`, die als Listener-Methode eingetragen wird, müssen zunächst alle Checkboxes ausgelesen werden und das Ergebnis in einer boolean-Variable festgehalten werden:

```
boolean e1 = cbBitebene1.isSelected();
```

Danach wird ein neues Bild schrittweise aufgebaut. Wenn eine Checkbox ausgewählt ist, dann wird die entsprechende Bitebene ermittelt und zum bisherigen Bild addiert.

Am Ende wird das Bild ggf. in ein Schwarz-Weiß-Bild umgewandelt.

5. Experimentiere, welche Bitebenen wie viel Information tragen.



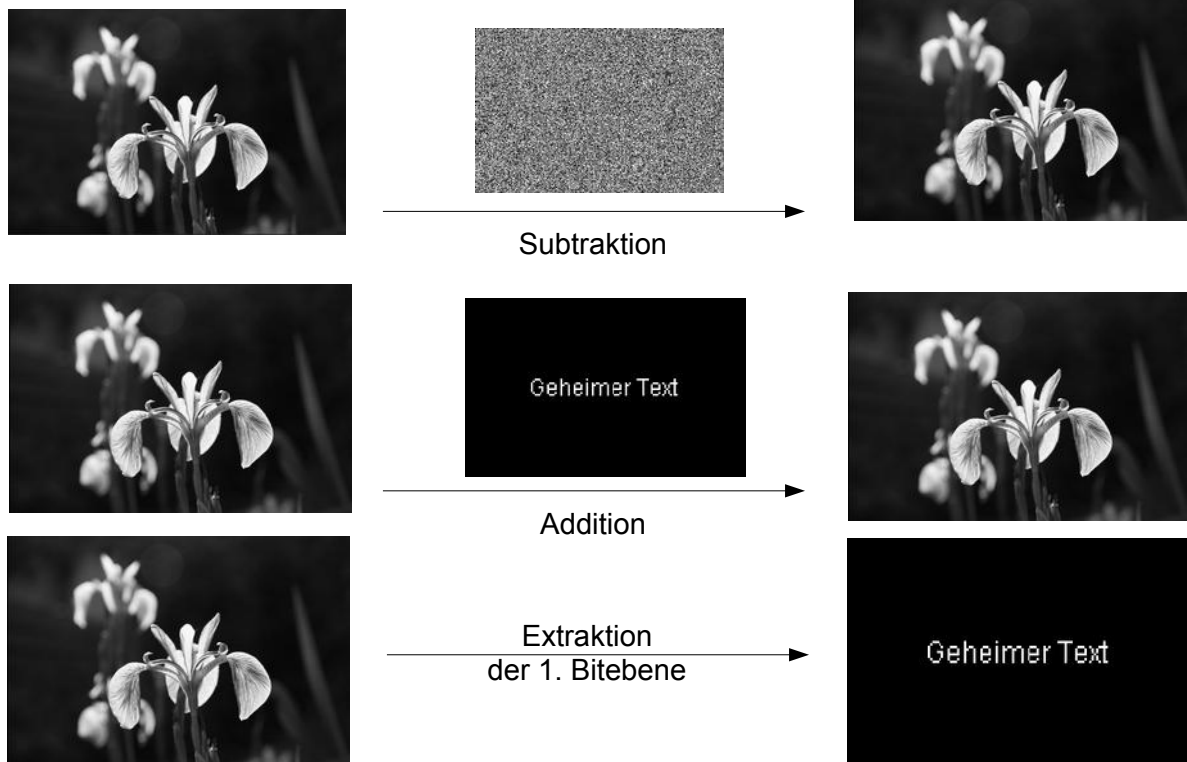


Steganographie


Man erkennt, dass die höherwertigen Bitebenen bestimmen, ob eine Region des Bildes hell oder dunkel ist. Die niederwertigen Bitebenen beeinflussen nur die feinen Strukturen. Dabei trägt die 1. Bit-Ebene so wenig bei, dass ihr Einfluss mit dem bloßen Auge nicht wahrnehmbar ist (sie sieht oben ohnehin nur nach Bildrauschen aus).

Dies kann man ausnutzen, um in dem Bild eine geheime Botschaft zu verstecken, ohne dass auffällt, dass das Bild überhaupt eine Nachricht enthält. Dies bezeichnet man als **Steganographie**. Dazu kann man z.B. die 1. Bit-Ebene eines Bildes gegen die 1. Bit-Ebene eines anderen Schwarz-Weiß-Bild austauschen, das die geheime Nachricht enthält. Extrahiert man aus dem veränderten Bild die 1. Bit-Ebene, erhält man die Nachricht zurück.

Steganographie darf allerdings nicht mit Verschlüsselung verwechselt werden, da hier kein Schlüssel verwendet wird. Jeder, der das Verfahren kennt, kann die Nachricht wieder extrahieren. Man kann dieses Verfahren z.B. auch verwenden, um ein Wasserzeichen in ein Bild einzubauen.



Aufgaben

6. Öffne ein beliebiges Bild. Extrahiere die 1. Bit-Ebene daraus. Subtrahiere diese vom Originalbild. Man erhält ein Bild das in der 1. Bit-Ebene nur Nullen hat.  Erstelle ein leeres schwarzes Bild (Background-Color = "000000") gleicher Größe. Schreibe mit der Methode `text(String s, int x, int y)` einen Text auf das Bild. Extrahiere die 1. Bit-Ebene daraus. Addiere diese zum manipulierten Originalbild. Speichere das Ergebnis ab. Dieses Bild enthält nun den versteckten Text. Um den Text zu extrahieren, wird erneut die erste Bildebene extrahiert und dann in ein Schwarz-Weiß-Bild umgewandelt (Grenze 1).



7. **Erstelle in der Klasse *Bitebenen* zwei Methoden** `public Picture steganographie_verstecken(Picture originalbild, String text)` **und** `public Picture steganographie_ermitteln(Picture originalbild)`, die diesen Prozess automatisieren.
8. **Füge dem Menü der Haupt-GUI einen neuen Menüpunkt "Steganographie" hinzu. Erstelle ein Optionsfenster mit einer Textarea, um den zu versteckenden Text einzugeben, und zwei Buttons zum Verstecken und Aufdecken eines versteckten Textes. Gib dem Textfeld eine `fxId` und lege Action-Methoden für die Buttons fest.**
- Lasse dieses Optionsfenster beim Aufruf des Menüpunktes anzeigen (Das aktuelle Bild wird hier ausnahmsweise nicht gesichert). Implementiere die beiden Action-Methoden, so dass sie `steganographie_verstecken` und `steganographie_ermitteln` aufrufen. Anschließend muss das Optionsfenster entfernt werden (vgl. Methode `bAusfuehren()`). Hinweis: Den Text einer `TextArea` bekommt man mit `getText()`.**

