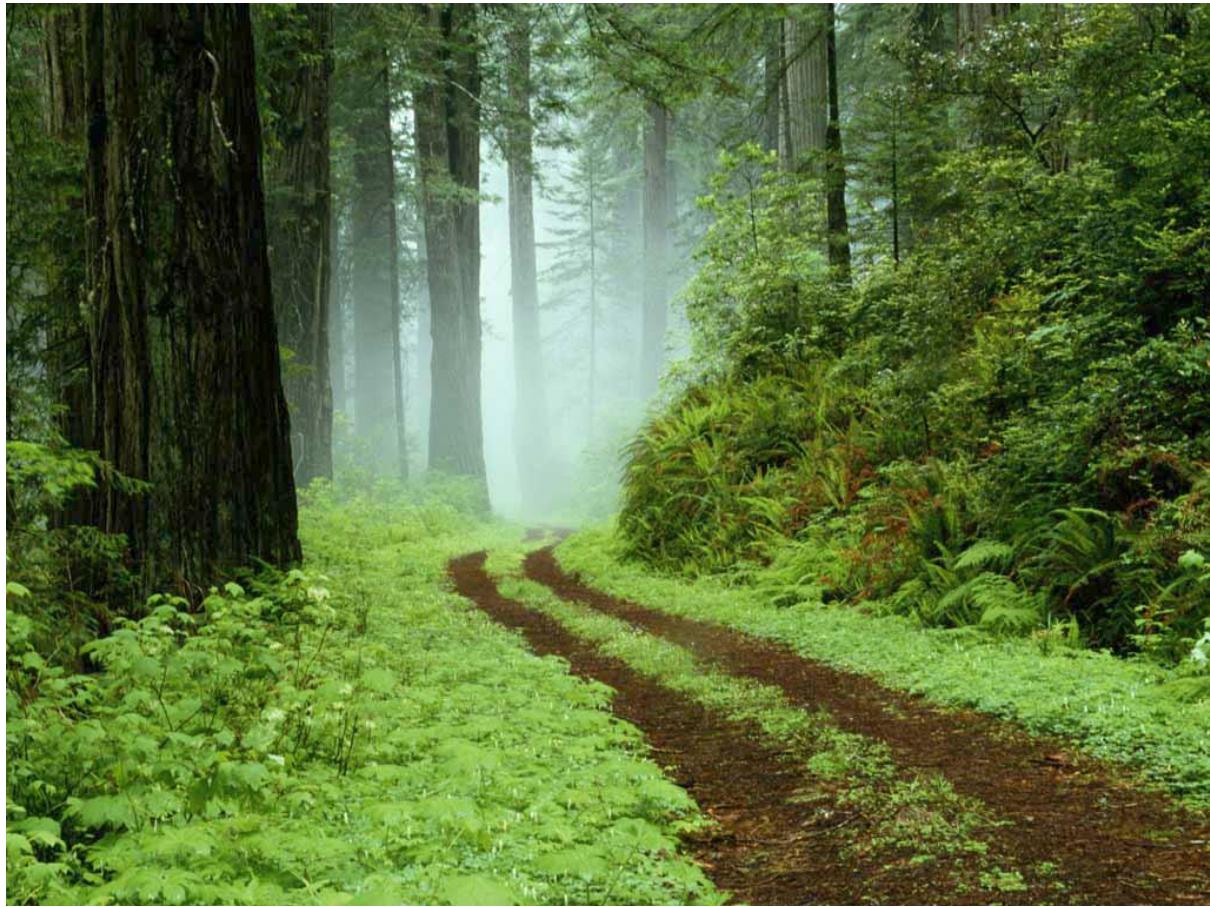
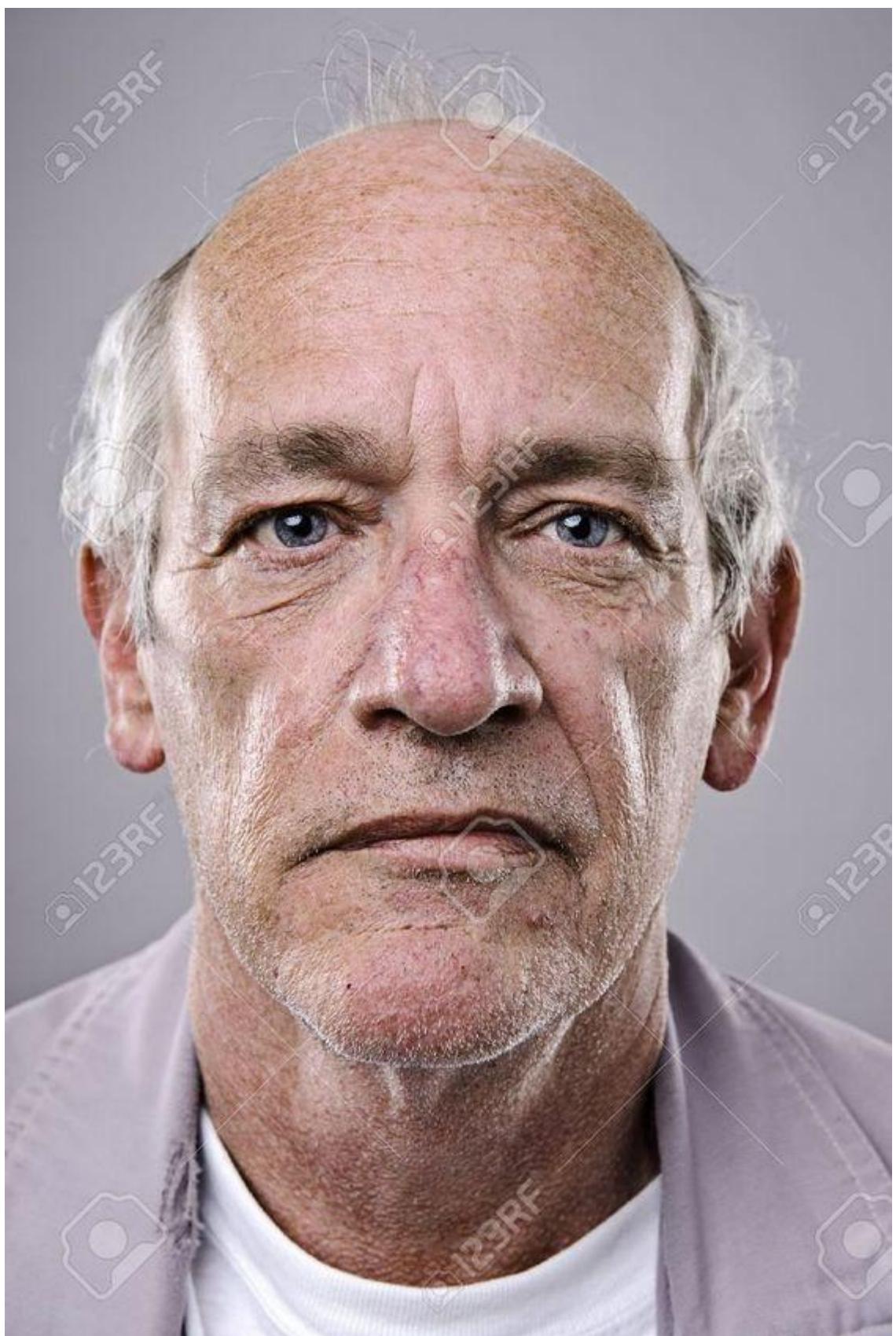


**Immagini Originali:**

Di seguito ci sono alcune delle immagini usate per testare i filtri







**Inverti [Facile]**

Questo filtro da come output l'inverso dell'immagine. È bastato, per ogni valore RGB di ogni pixel, calcolare  $1 - R$ ,  $1 - G$ ,  $1 - B$



### Blur [Facile]

A differenza del gaussian Blur (che vedremo in seguito) questo filtro calcola per ogni pixel, la media dei colori di tutti i pixel che rientrano nel raggio  $k$  (nelle foto  $k= 3, 10, 20$ ) partendo dalla posizione del pixel e la media sarà il nuovo colore del pixel





### Cut [Facile]

ritaglia (anche se sarebbe più corretto dire che annerisce) i pixel che non rientrano in  $((x-X/2)^2/(X/2)^2 + (y-Y/2)^2/(Y/2)^2)$ .



### EdgeDetection [Medio]

(Riferimento: <https://old.cescg.org/CESCG97/boros/>)

questo filtro serve per rilevare i bordi dei vari elementi nella foto. Per ottenerlo ho usato delle convolution masks. Solitamente questo filtro è accompagnato da un lievissimo blur per ridurre il rumore (non è stato applicato in nessuna delle immagini sotto)

- Mi scorro l'immagine e mi tengo traccia, per ogni pixel, della somma dei vari canali (dove ogni elemento della somma è stato moltiplicato per il relativo elemento della maschera).
- Mi tengo un contatore
- per ogni canale, alla fine della maschera, sommo eventualmente un bias e/o divido per un koef ed infine setto il pixel

Sia  $M$  una maschera  $3 \times 3$  allora il nuovo pixel avrà valore:

```

New[x,y] = Old[x-1,y-1] * M[1,1] +
           Old[x,y-1]   * M[2,1] +
           Old[x+1,y-1] * M[3,1] +
           Old[x-1,y ]  * M[1,2] +
           Old[x ,y ]   * M[2,2] +
           Old[x+1,y ]  * M[3,2] +
           Old[x-1,y+1] * M[1,3] +
           Old[x ,y+1]   * M[2,3] +
           Old[x+1,y+1] * M[3,3]

```

Sono stati implementati 3 livelli di edge detection:

<b>Heavy edge detection</b>	1	-2	1
	-2	4	-2
	1	-2	1

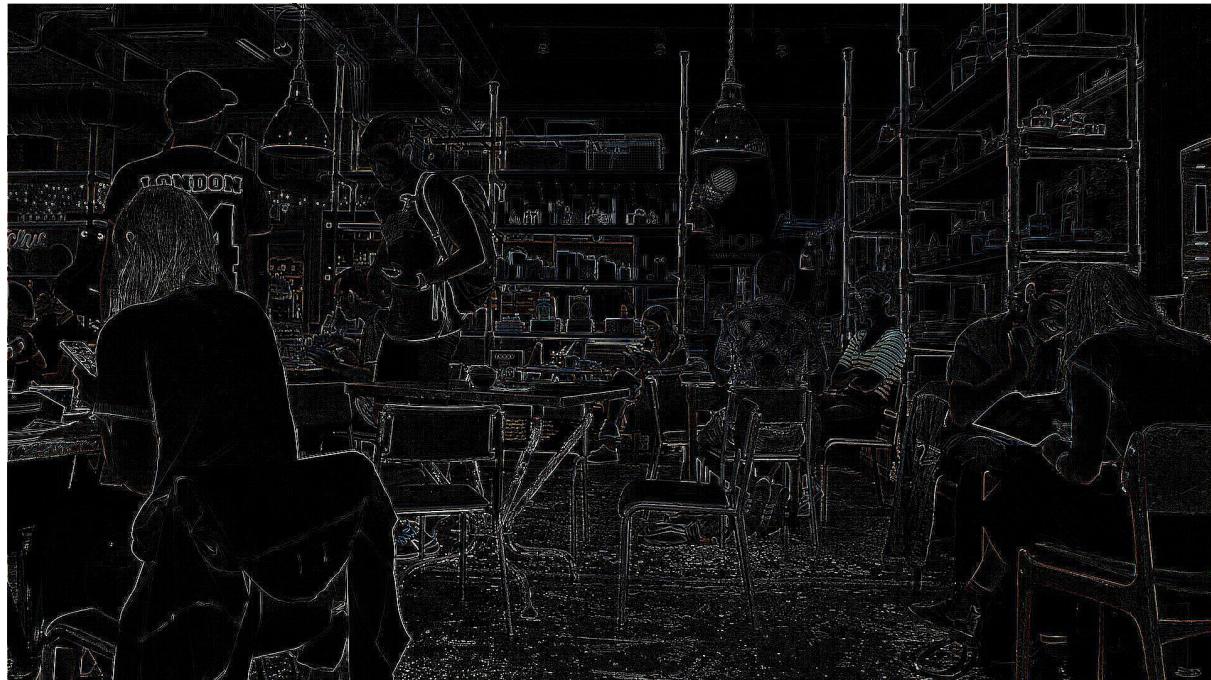


-1      -1      -1

### Medium edge detection

-1      8      -1

-1      -1      -1

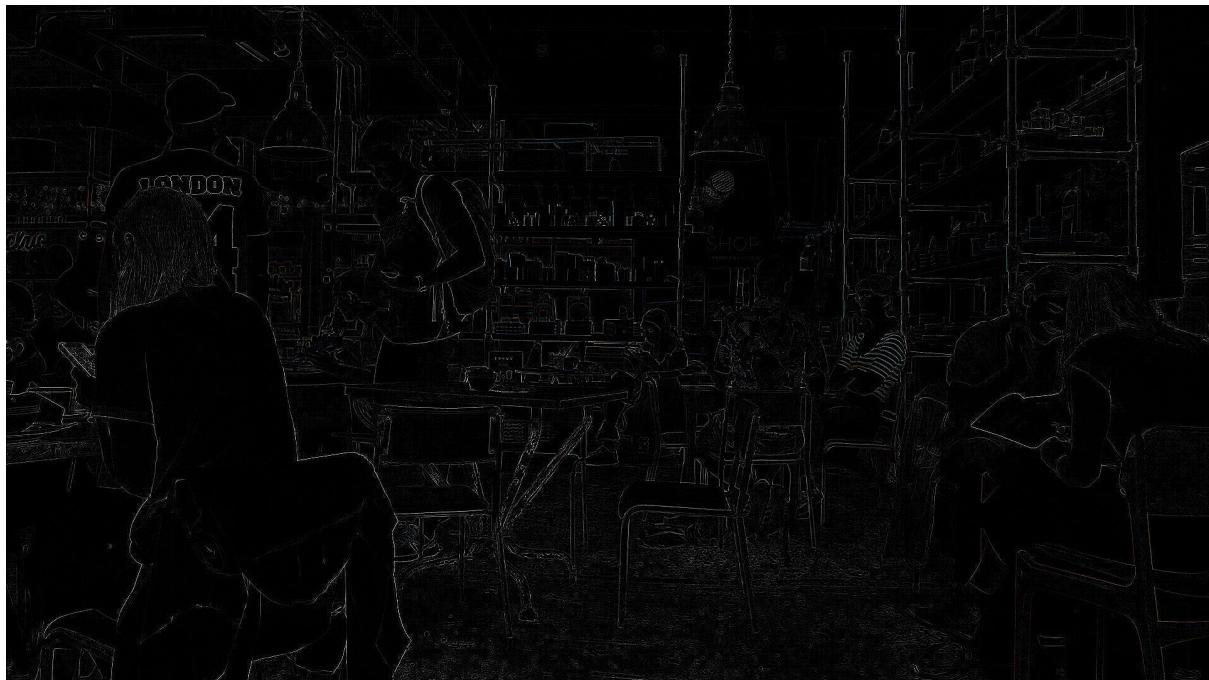


0 1 0

## Light edge detection

1      -4      1

0      1      0



## Emboss [Medio]

Serve per fare un rilievo dell'immagine. Stesso principio del edge detection. Uso una mask per ottenere un rilievo della foto.

- Mi scorro l'immagine e mi tengo traccia, per ogni pixel, della somma dei vari canali (dove ogni elemento della somma è stato moltiplicato per il relativo elemento della maschera).
- Mi tengo un contatore
- per ogni canale, alla fine della maschera, sommo eventualmente un bias e/o divido per un koef ed infine setto il pixel

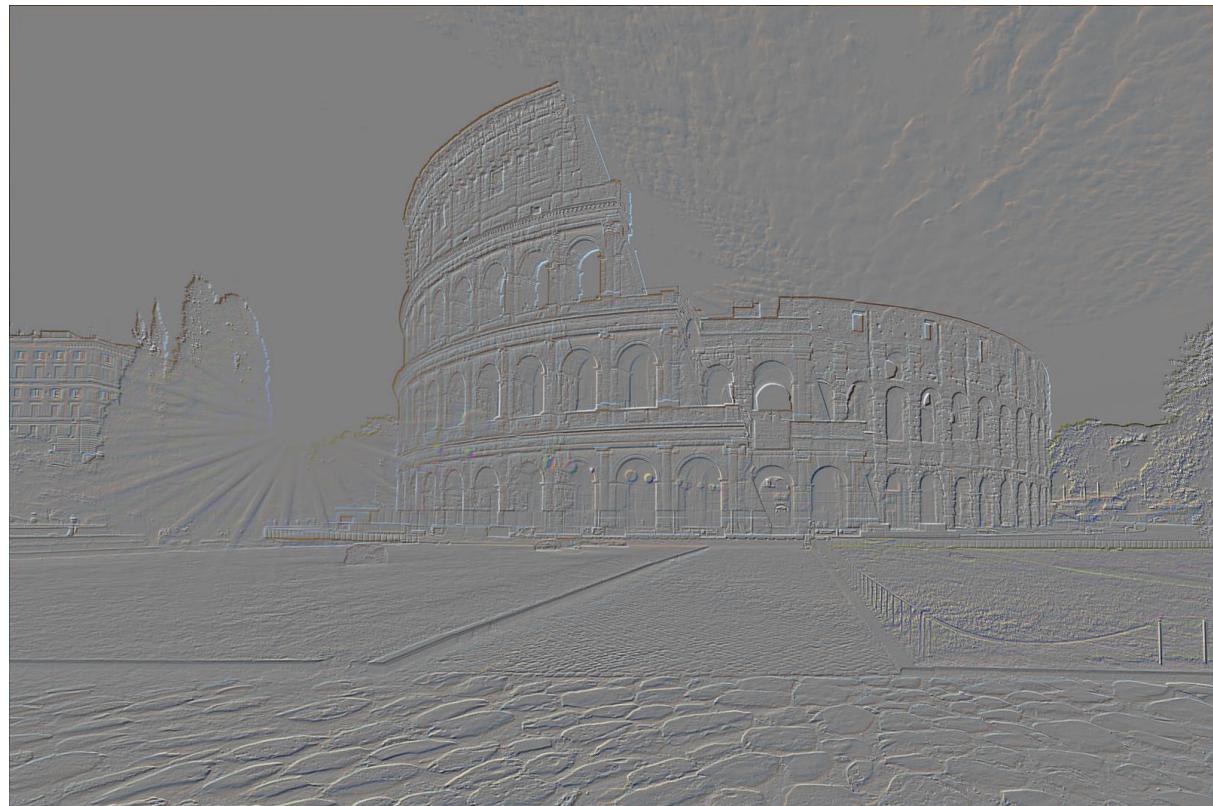
La maschera usata è;

1      0      0

Il bias per questo filtro è  $Z/2$  (e un koef = 1) dove Z è la massima intensità (1 nel nostro caso).

$\text{New}[x,y] = \text{New}[x,y]/\text{koef} + \text{Bias}$

0 0 0



### EnhancedDetails [Medio]

Filtro molto leggero che permette di mettere in risalto alcuni dettagli (nella foto allegata si nota la differenza dalla foto originale facendo uno zoom sulla pianta, sul quadro o sui libri). Oltre alla mask è stato usato anche un koef = 6

$\text{New}[x,y] = \text{New}[x,y]/\text{koef} + \text{Bias}$

- Mi scorro l'immagine e mi tengo traccia, per ogni pixel, della somma dei vari canali (dove ogni elemento della somma è stato moltiplicato per il relativo elemento della maschera).
- Mi tengo un contatore
- per ogni canale, alla fine della maschera, sommo eventualmente un bias e/o divido per un koef ed infine setto il pixel

La mask usata è:

0 -1 0

-1 10 -1

0 -1 0



### trueAnaglyph [Medio]

(Riferimento: [http://www.3dtv.at/knowhow/anaglyphcomparison\\_en.aspx](http://www.3dtv.at/knowhow/anaglyphcomparison_en.aspx))

Filtro che serve per ottenere un effetto 3d, ogni pixel (ignorando l'alpha channel) è dato da

$$\begin{pmatrix} r_a \\ g_a \\ b_a \end{pmatrix} = \begin{pmatrix} 0,299 & 0,587 & 0,114 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0,299 & 0,587 & 0,114 \end{pmatrix} \cdot \begin{pmatrix} r_2 \\ g_2 \\ b_2 \end{pmatrix}$$

dove  $r_1g_1b_1$  sono i colori del primo pixel mentre  $r_2g_2b_2$  sono i colori del secondo pixel. Il secondo pixel è stato preso a  $k$  posizioni dal primo pixel (se quelle  $i + k \Rightarrow \text{image.width}$ , prendo i valori  $r_1g_1b_1$ )



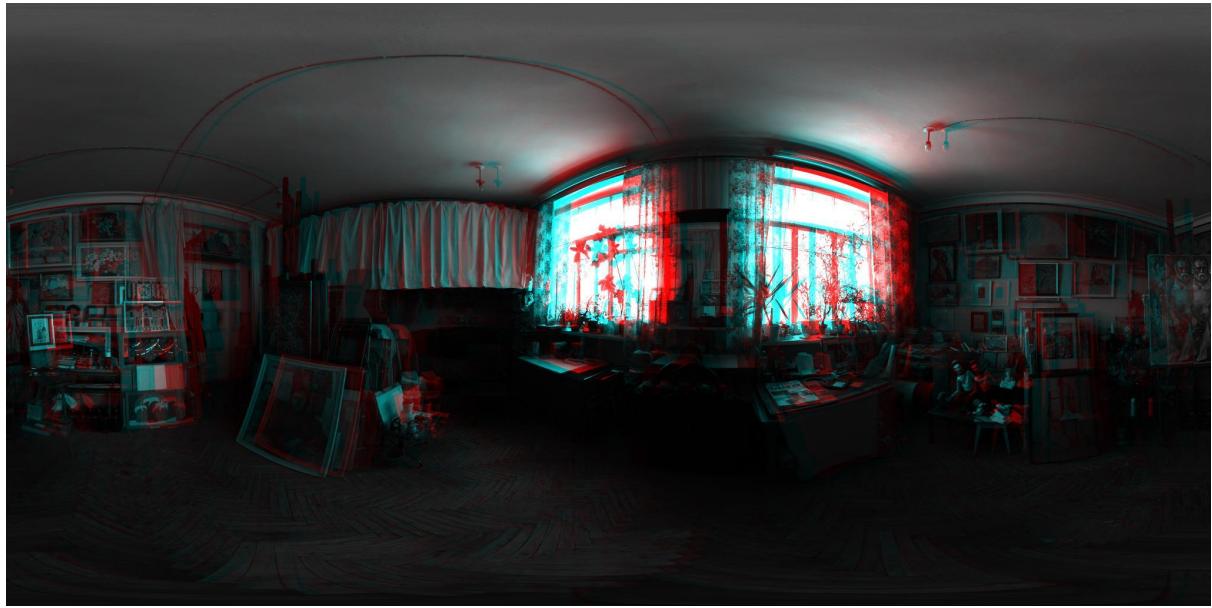
### grayAnaglyph [Medio]

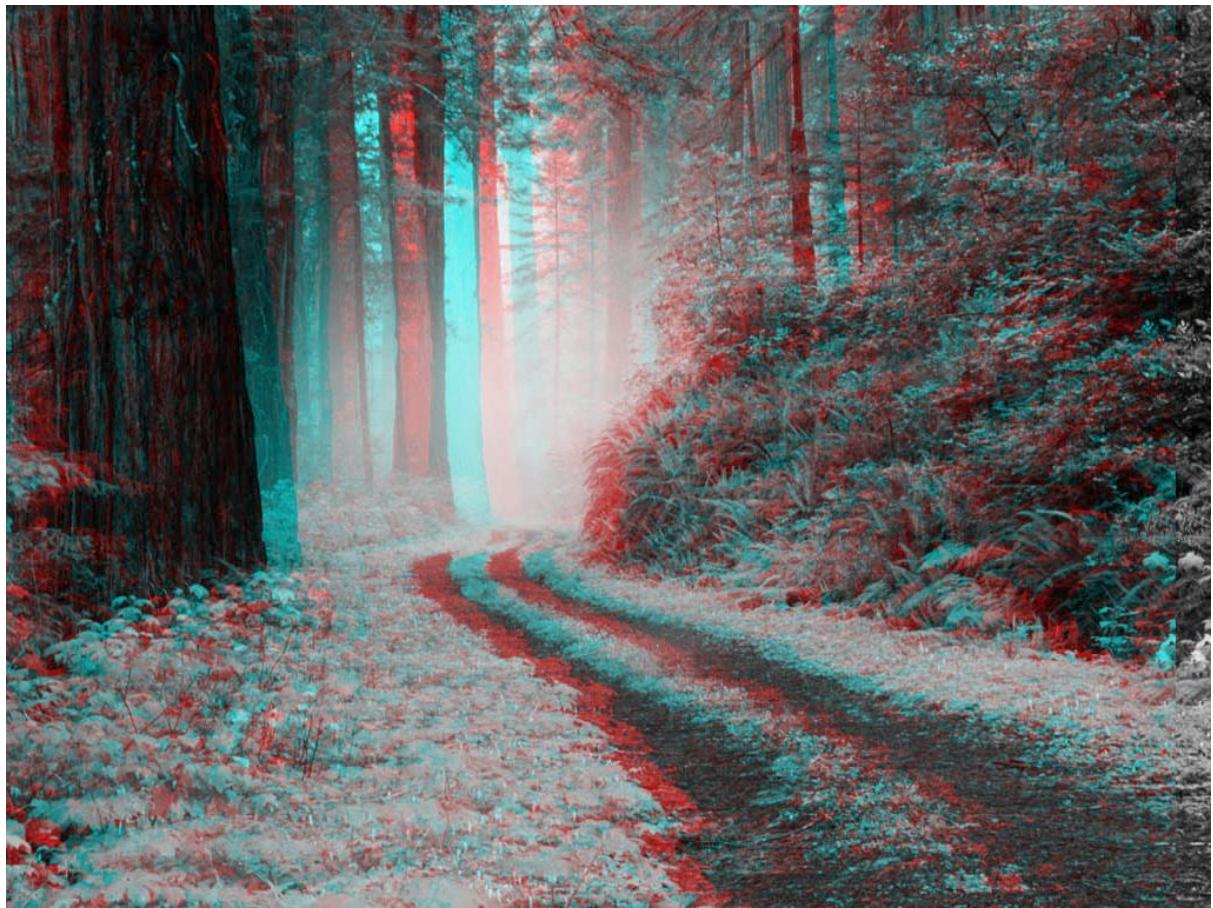
(Riferimento: [http://www.3dtv.at/knowhow/anaglyphcomparison\\_en.aspx](http://www.3dtv.at/knowhow/anaglyphcomparison_en.aspx))

Filtro che serve per ottenere un effetto 3d, ogni pixel (ignorando l'alpha channel) è dato da

$$\begin{pmatrix} r_a \\ g_a \\ b_a \end{pmatrix} = \begin{pmatrix} 0,299 & 0,587 & 0,114 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0,299 & 0,587 & 0,114 \\ 0,299 & 0,587 & 0,114 \end{pmatrix} \cdot \begin{pmatrix} r_2 \\ g_2 \\ b_2 \end{pmatrix}$$

dove r1g1b1 sono i colori del primo pixel mentre r2g2b2 sono i colori del secondo pixel. Il secondo pixel è stato preso a k posizioni dal primo pixel (se quelle i + k => image.width, prendo i valori r1g1b1)





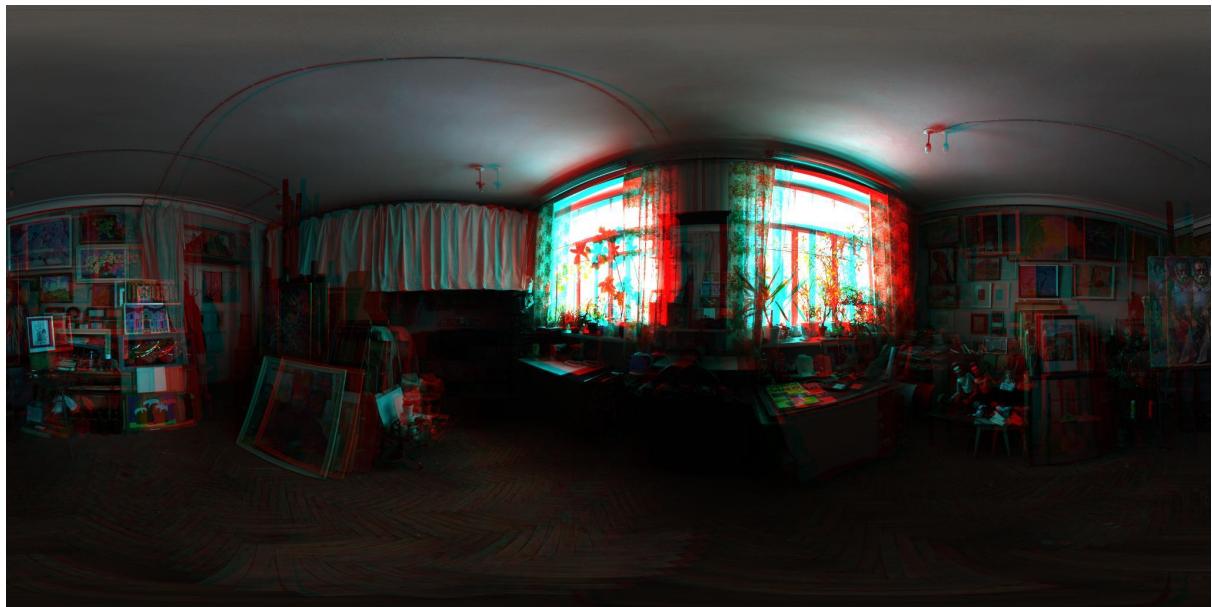
### colorAnaglyph [Medio]

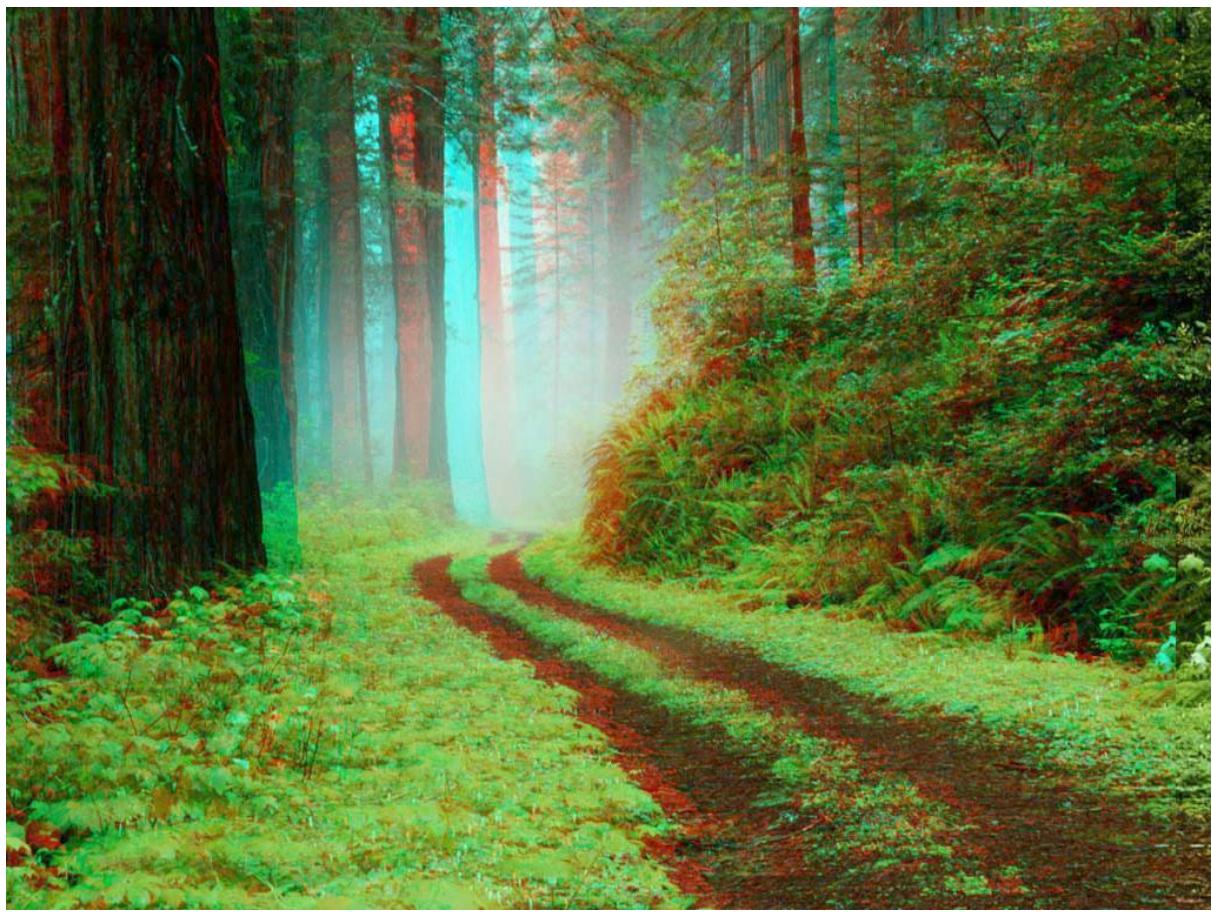
(Riferimento: [http://www.3dtv.at/knowhow/anaglyphcomparison\\_en.aspx](http://www.3dtv.at/knowhow/anaglyphcomparison_en.aspx))

Filtro che serve per ottenere un effetto 3d, ogni pixel (ignorando l'alpha channel) è dato da

$$\begin{pmatrix} r_a \\ g_a \\ b_a \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r_2 \\ g_2 \\ b_2 \end{pmatrix}$$

dove r1g1b1 sono i colori del primo pixel mentre r2g2b2 sono i colori del secondo pixel. Il secondo pixel è stato preso a k posizioni dal primo pixel (se quelle i + k => image.width, prendo i valori r1g1b1)





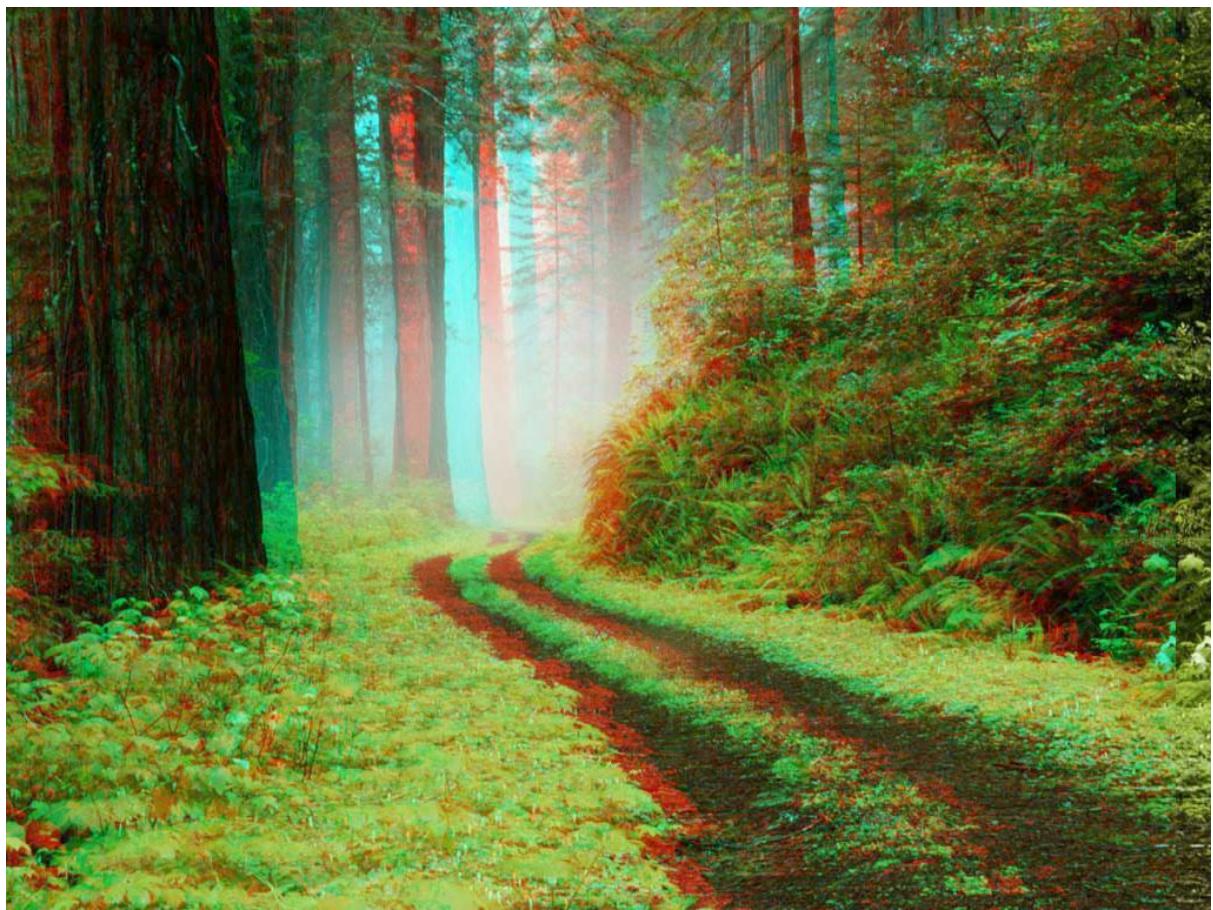
### halfColorAnaglyph [Medio]

(Riferimento: [http://www.3dtv.at/knowhow/anaglyphcomparison\\_en.aspx](http://www.3dtv.at/knowhow/anaglyphcomparison_en.aspx))

Filtro che serve per ottenere un effetto 3d, ogni pixel (ignorando l'alpha channel) è dato da

$$\begin{pmatrix} r_a \\ g_a \\ b_a \end{pmatrix} = \begin{pmatrix} 0,299 & 0,587 & 0,114 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r_2 \\ g_2 \\ b_2 \end{pmatrix}$$

dove r1g1b1 sono i colori del primo pixel mentre r2g2b2 sono i colori del secondo pixel. Il secondo pixel è stato preso a k posizioni dal primo pixel (se quelle i + k => image.width, prendo i valori r1g1b1)



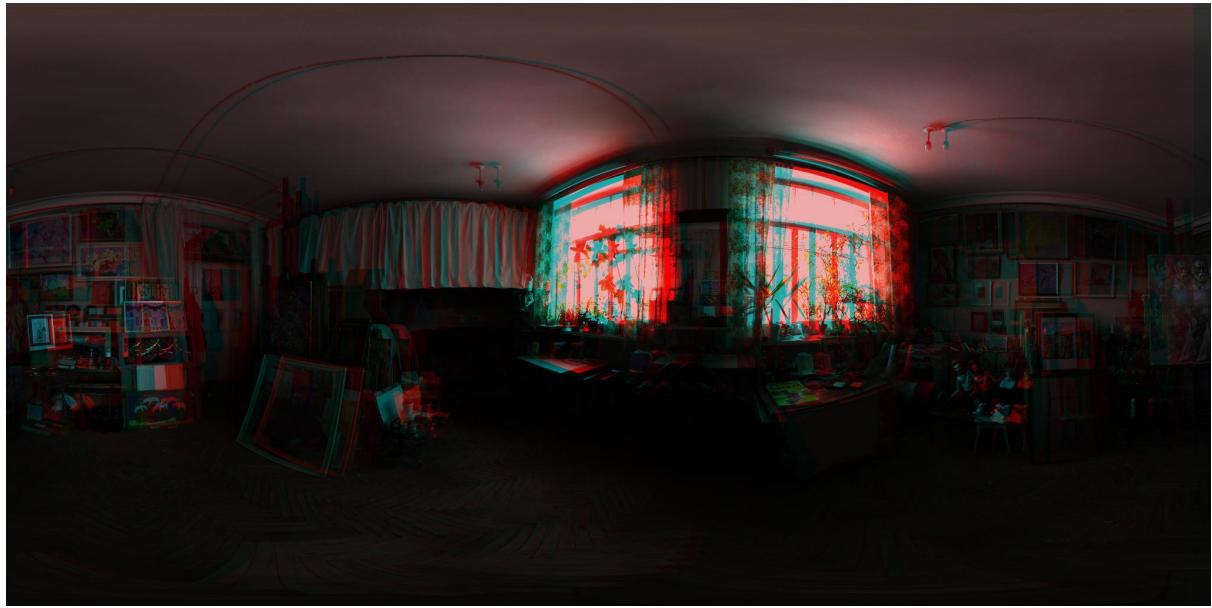
### **optimizedColorAnaglyph [Medio]**

(Riferimento: [http://www.3dtv.at/knowhow/anaglyphcomparison\\_en.aspx](http://www.3dtv.at/knowhow/anaglyphcomparison_en.aspx))

Filtro che serve per ottenere un effetto 3d, ogni pixel (ignorando l'alpha channel) è dato da

$$\begin{pmatrix} r_a \\ g_a \\ b_a \end{pmatrix} = \begin{pmatrix} 0 & 0,7 & 0,3 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_2 \\ g_2 \\ b_2 \end{pmatrix}$$

dove r1g1b1 sono i colori del primo pixel mentre r2g2b2 sono i colori del secondo pixel. Il secondo pixel è stato preso a k posizioni dal primo pixel (se quelle i + k => image.width, prendo i valori r1g1b1)





### Soft [Medio]

Anche questo filtro è molto leggero e permette di togliere i dettagli in eccesso come ad esempio i pori. Anche in questo caso sono state usate delle maschere e un koeff.

- Mi scorro l'immagine e mi tengo traccia, per ogni pixel, della somma dei vari canali (dove ogni elemento della somma è stato moltiplicato per il relativo elemento della maschera).
- Mi tengo un contatore
- per ogni canale, alla fine della maschera, sommo eventualmente un bias e/o divido per un koef ed infine setto il pixel

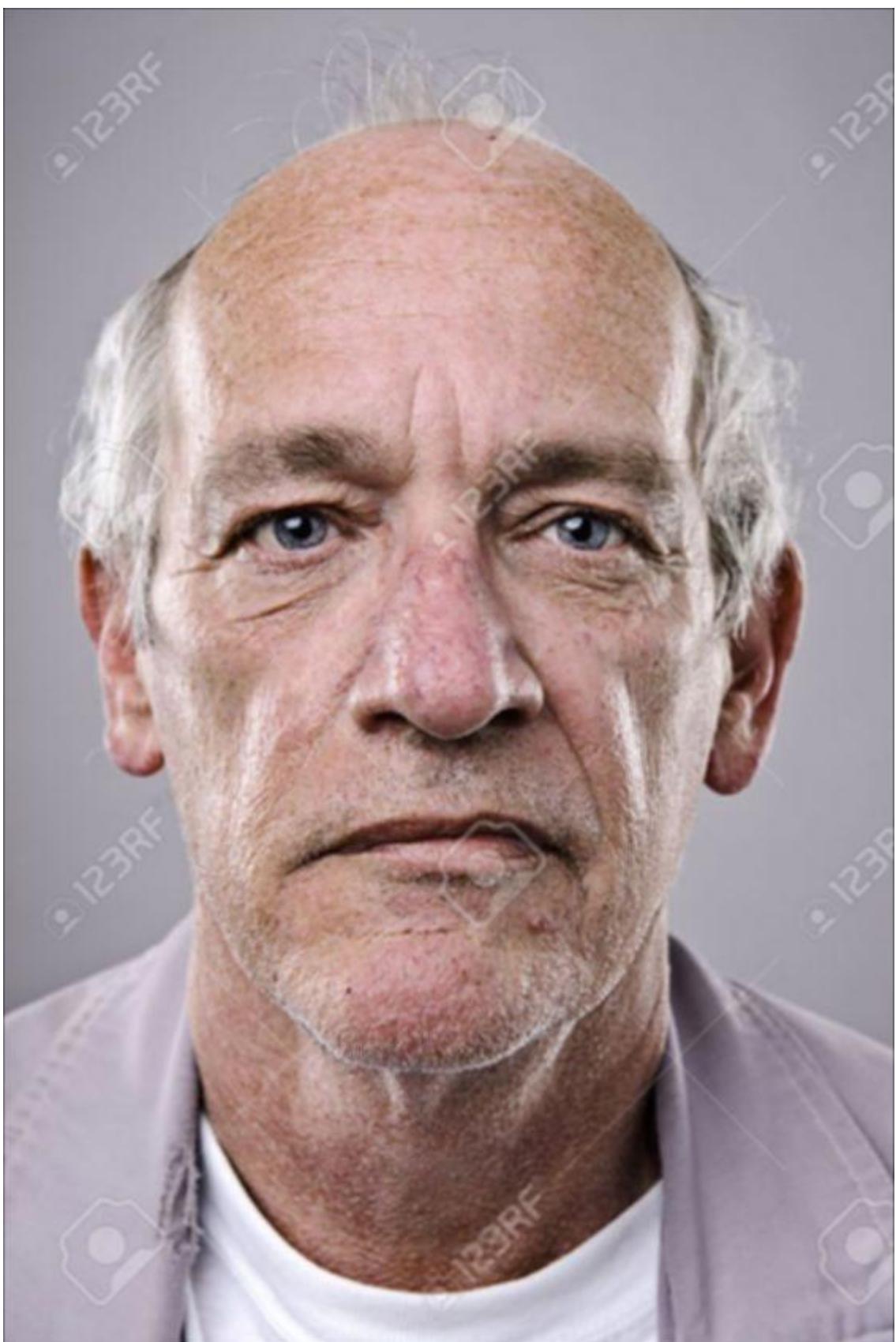
### • Heavy soften filter

11 11 11

koef = 99

11 11 11

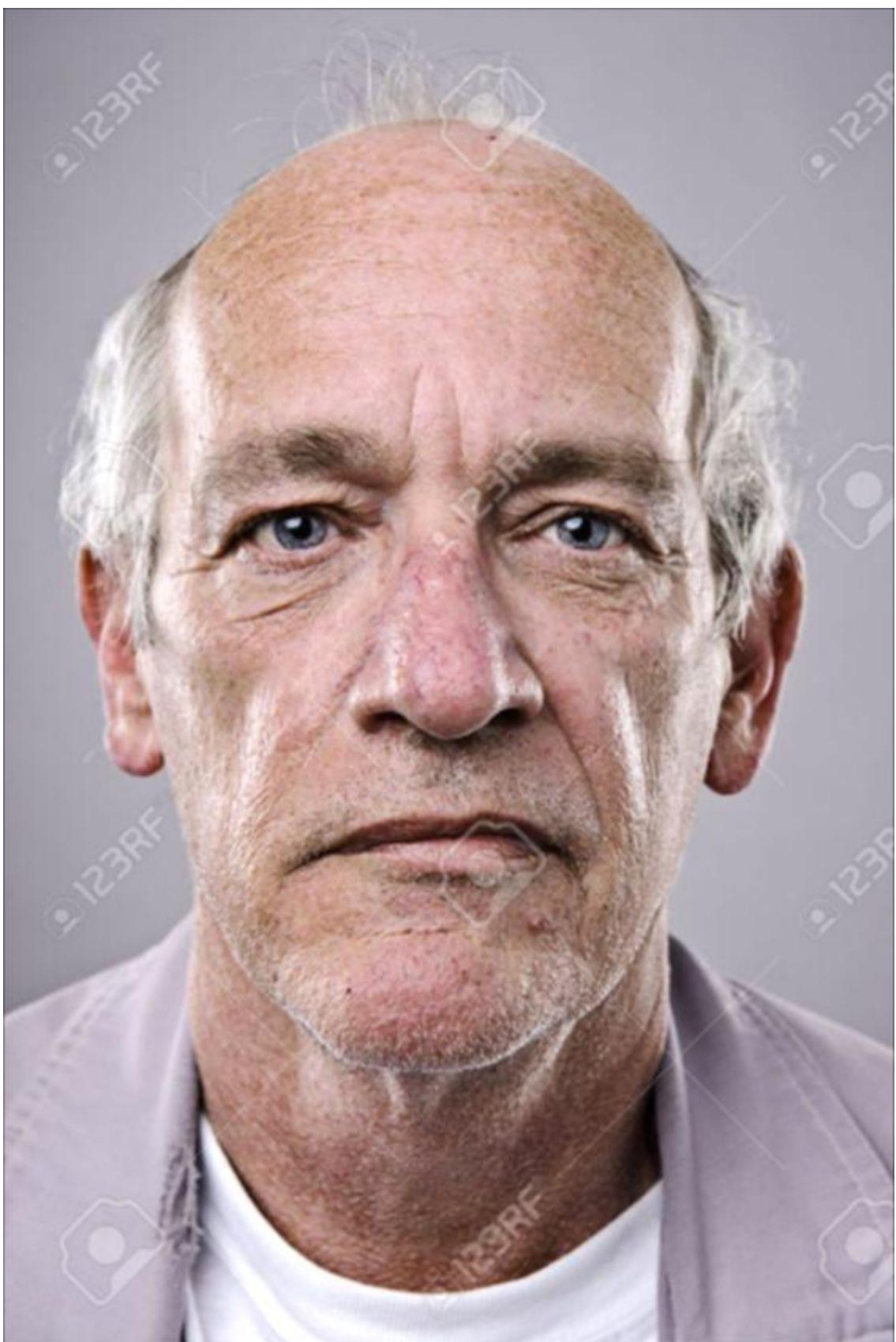
11 11 11



• **Medium soften filter**      10      10      10

**koeff = 100**      10      20      10

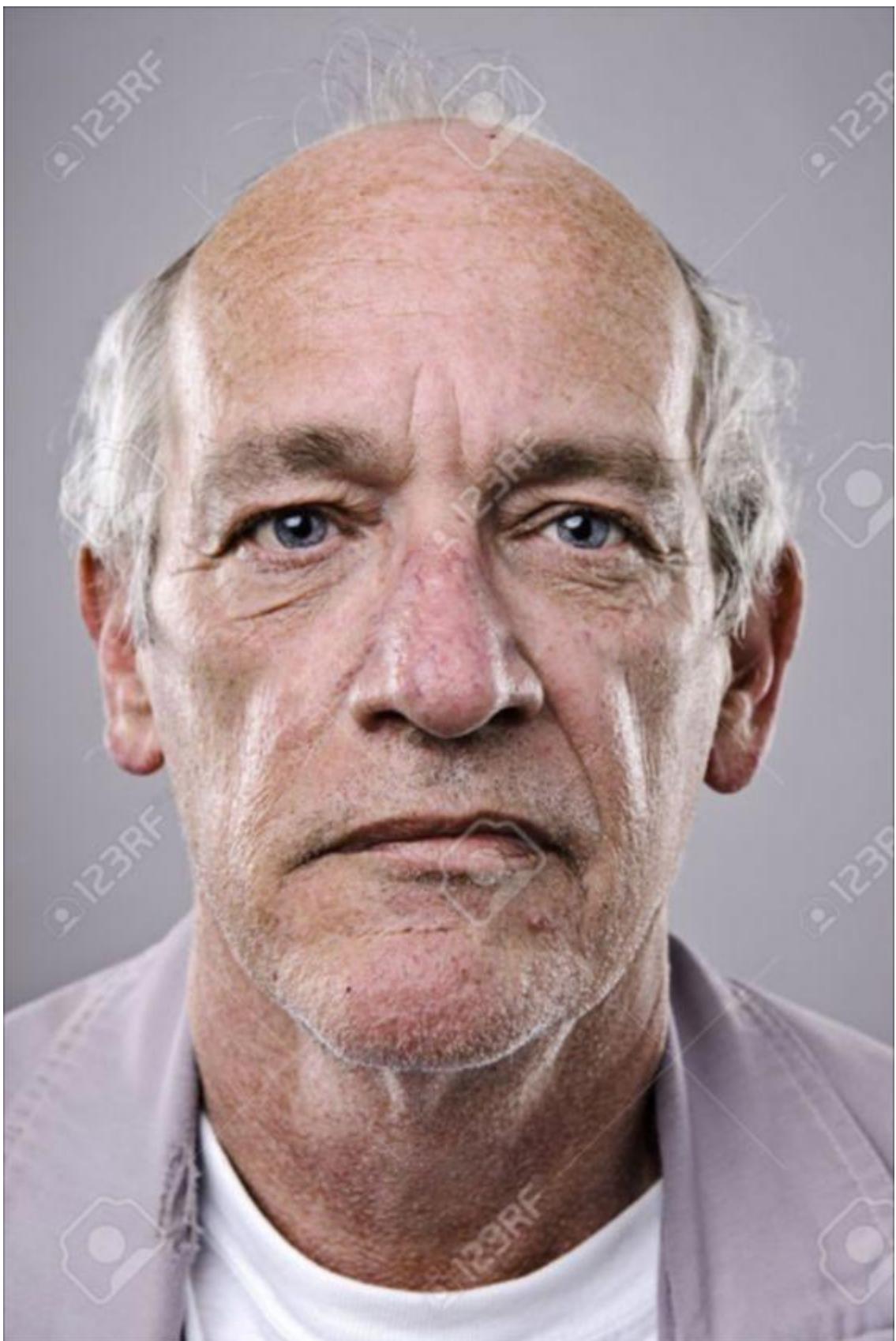
10      10      10



• Light soften filter      6    12    6

koeff = 97      12    25    12

6    12    6



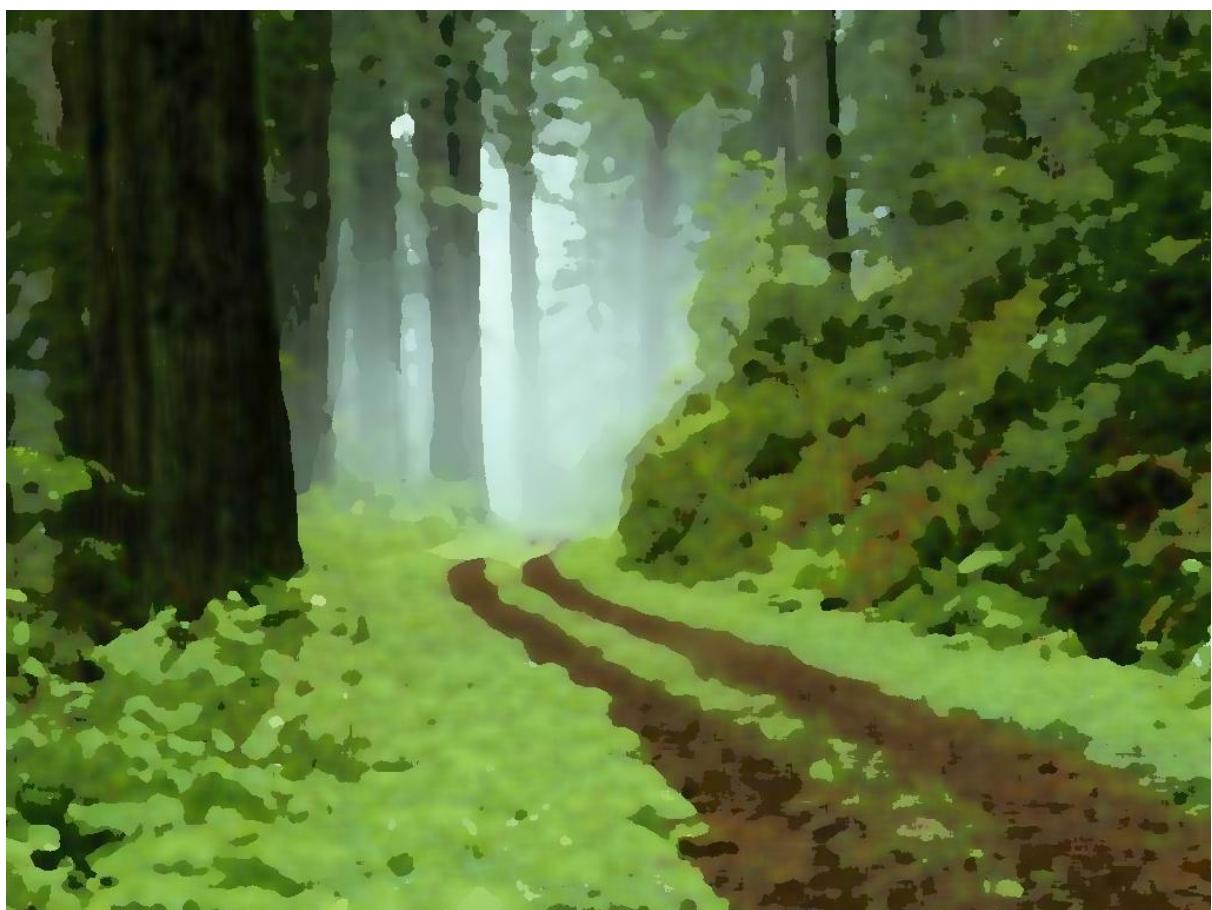
## **Oil [Medio]**

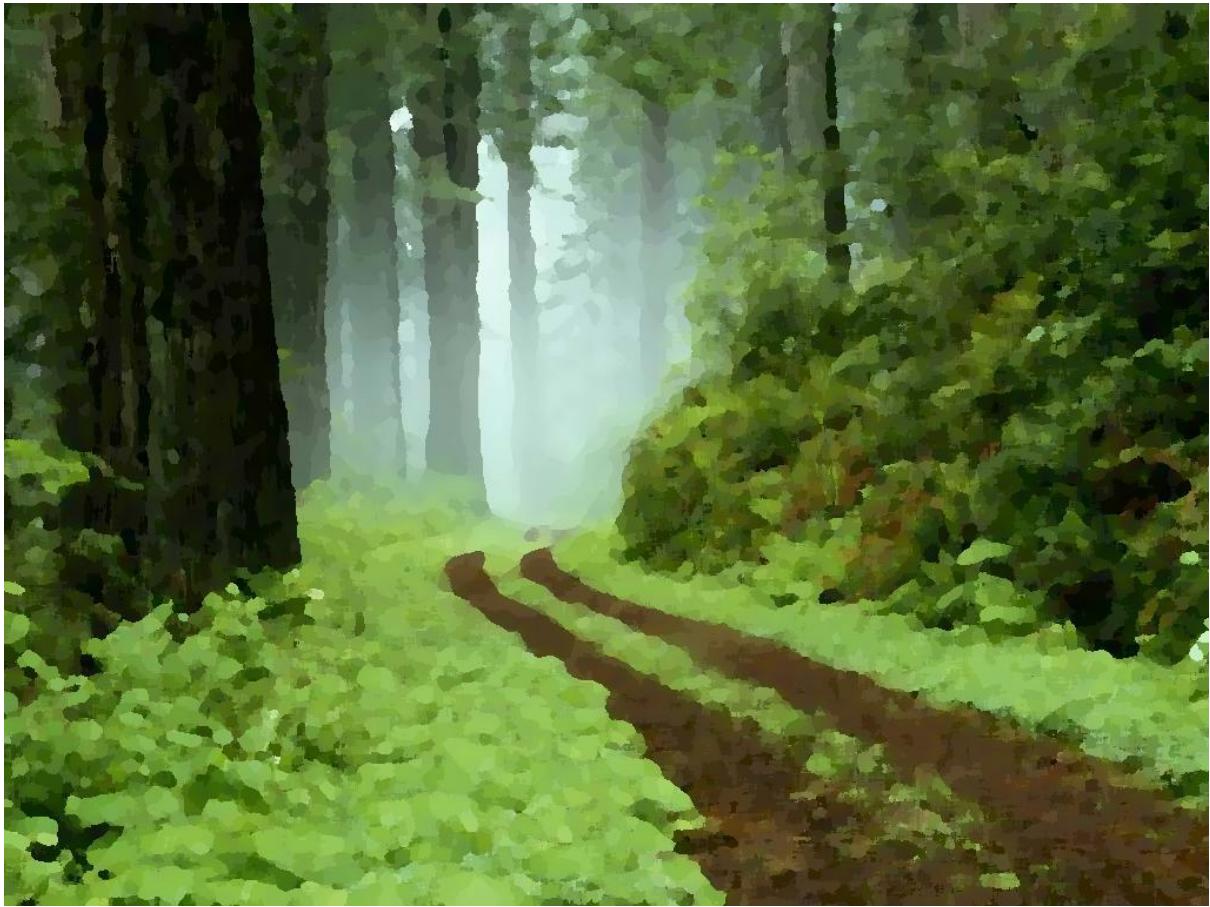
(Riferimento: <http://supercomputingblog.com/graphics/oil-painting-algorithm/>)

Filtro che permette di dare in out un'immagine che sembra un dipinto. Prende come parametro intensityLevel che più aumenta e più il dipinto ottiene dettagli.

Ho usato anche la libreria standard Map per poter gestire in maniera veloce, semplice e leggibile (per me) i vari valori delle diverse intensità.

- Mi creo delle mappe per salvarmi dei valori RGB e per salvarmi un contatore per ogni intensità.
- I livelli di intesità sono definiti in params.
- Per ogni pixel, mi calcolo un max intesità
- L'intensità corrente è definita come  $((((r+g+b)/3)*intensityLevels)/255)$  e se è maggiore dell'intensità massima ottenuta allora me la salvo come max intesità.
- Sommo 1 al valore che si trova nella mappa dei count[curlIntensity] e somma i valori RGB ai rispettivi dizionari
- I valori RGB del pixel saranno definiti da diz<channel>[keyMax] / ValueMax dove keyMax è la chiave della mappa di count maggiore di tutti e valueMax sarà il valore del count più grosso.





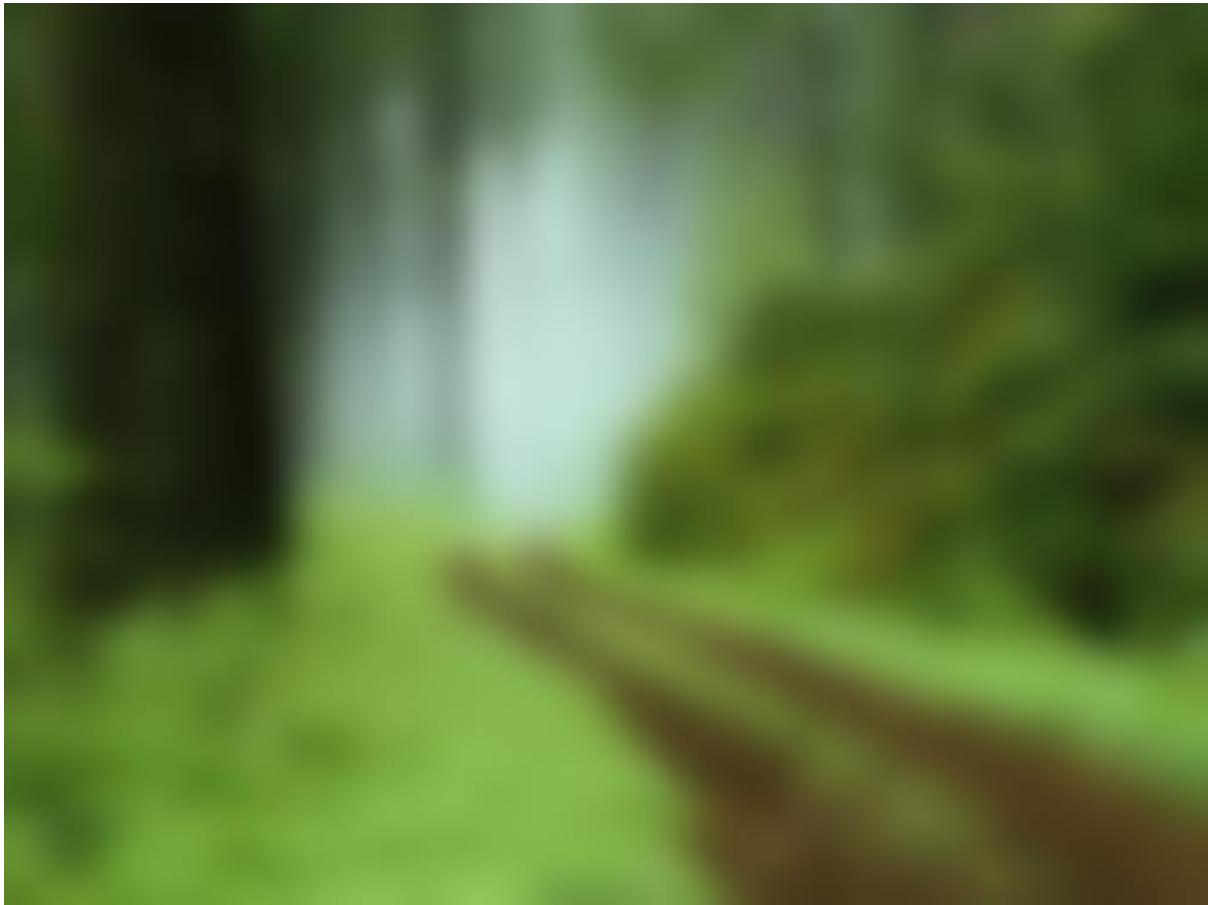
### gaussianBlur

(Riferimento: [https://en.wikipedia.org/wiki/Gaussian\\_blur](https://en.wikipedia.org/wiki/Gaussian_blur))

A differenza del blur mostrato all'inizio del documento, qui anziché fare la media secca tra tutti i pixel che capitano nel mio raggio, faccio una media pesata e calcolo il tutto con la funzione di gauss che ho manipolato con sigma.

La funzione di gauss è:  $\exp(-(pow((float)i, 2) / (2 * pow(sigma, 2)))) / \sqrt{2 * pi * pow(sigma, 2)}$ ;

- Mi calcolo i valori che ottengo con la funzione di gauss e dopo li normalizzo
-



### Sketch [Facile]

(Riferimento: <https://www.shadertoy.com/view/7scSRX>)

Filtro che permette di creare un disegno a partire da un'immagine in input.

Mi creo un vec2f con le size dell'immagine, un altro vec2f con le coordinate attuali del pixel che sto considerando e mi creo un altro vec2f che non è altro che la divisione tra i due pixel.  
Infine faccio

```
silhouette = vec4f{1, 1, 1} * (smoothstep(0., .8, 1. - pow(r, 6))); col = lerp(c * silhouette,  
silhouette , float(sin(params.boh * .8) * .5 + .5));  
e il mio pixel non sarà altro che col
```





<http://supercomputingblog.com/openmp/stained-glass-algorithm/>

### CRTTV [FACILE]

(Riferimento: <https://www.shadertoy.com/view/Nd3SWX>)

Filtro che serve per emulare le vecchie TV CRT. Crea un leggero effetto sfarfallio

