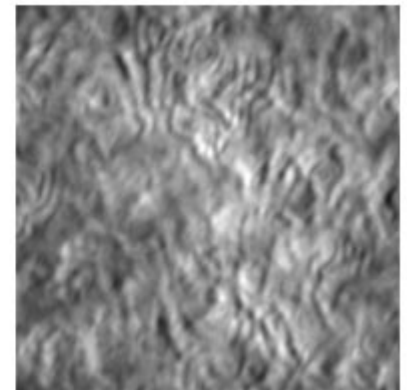
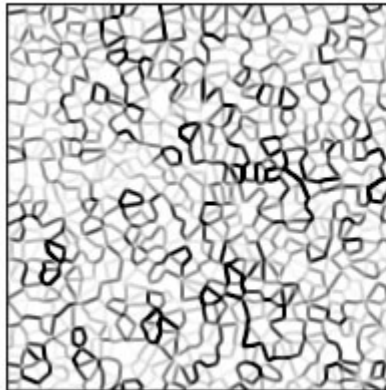


# Parametrische Texturgeneration

---

inf.misc

19.05.2004



# Überblick

- Wozu ? / Grundidee
- Erzeugende Algorithmen
- Verändernde Algorithmen/Filter
- Details zur Implementierung
- Weiterführendes/Ideen
- Fragen, Kommentare ?

# Wozu ?

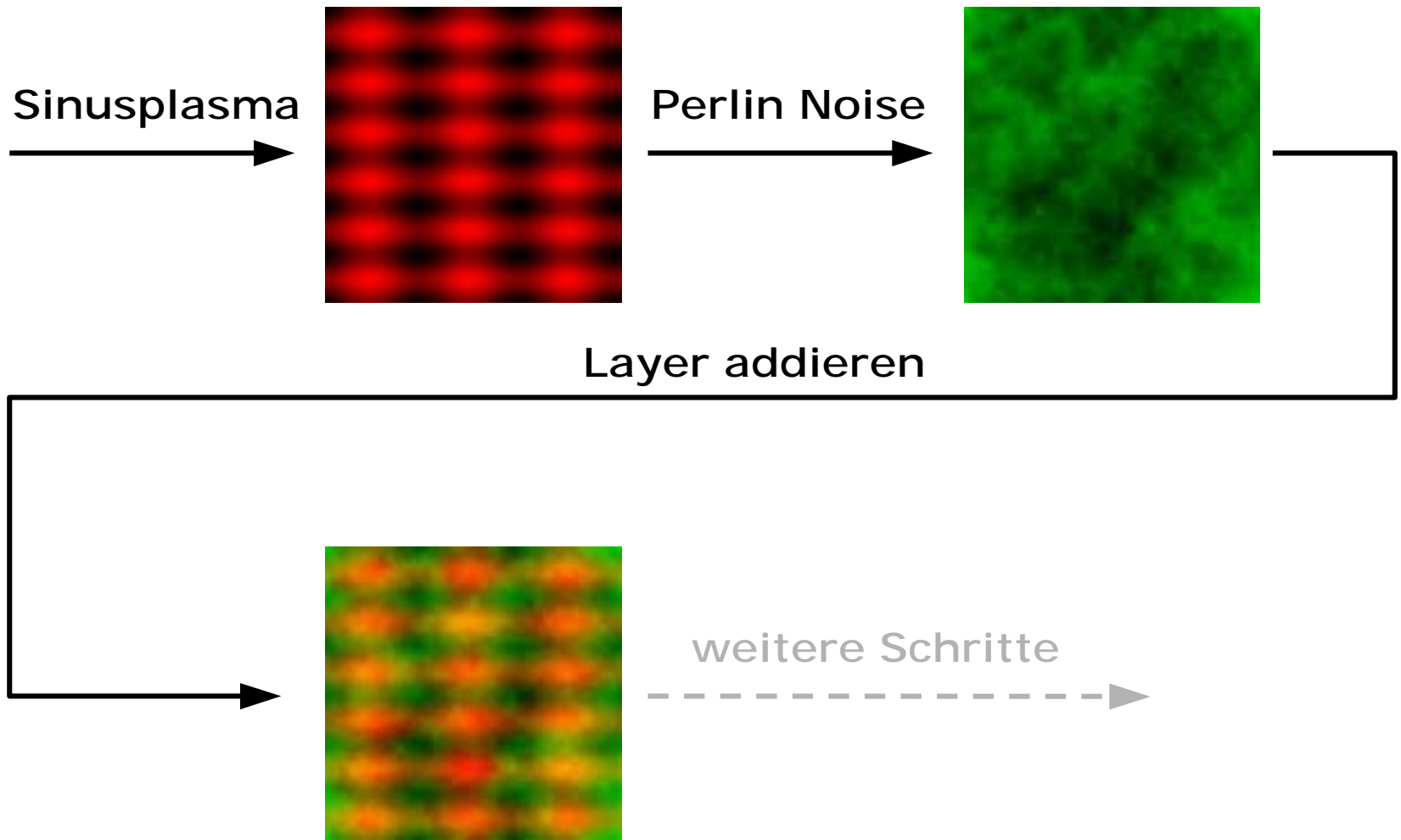
- Umgebung stellt nur wenig Speicher für Daten zur Verfügung
- Animierte Texturen; z.B. Wolken
- Grundlage für handverfeinerte Texturen
- Anwendung soll kompakt sein und ohne viel externe Daten auskommen

# Grundidee


- Einfache Algorithmen verwenden, um Pixel-  
daten zu erzeugen
- Layer modifizieren und kombinieren
- Nur Funktionsparameter statt Pixeldaten  
speichern

→ Mehr Code, weniger Daten

# Vereinfachtes Beispiel

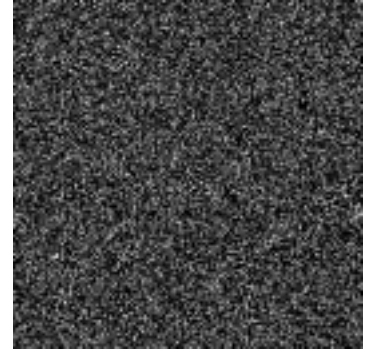


# Zu überdenken

- Texturgröße ? [256, 256]
- Tileable ? 
- Parameter-Datentyp ? BYTE
- Konzept ausarbeiten !

# #1 - Noise

Idee: Zufallswerte in gegebener  
Menge



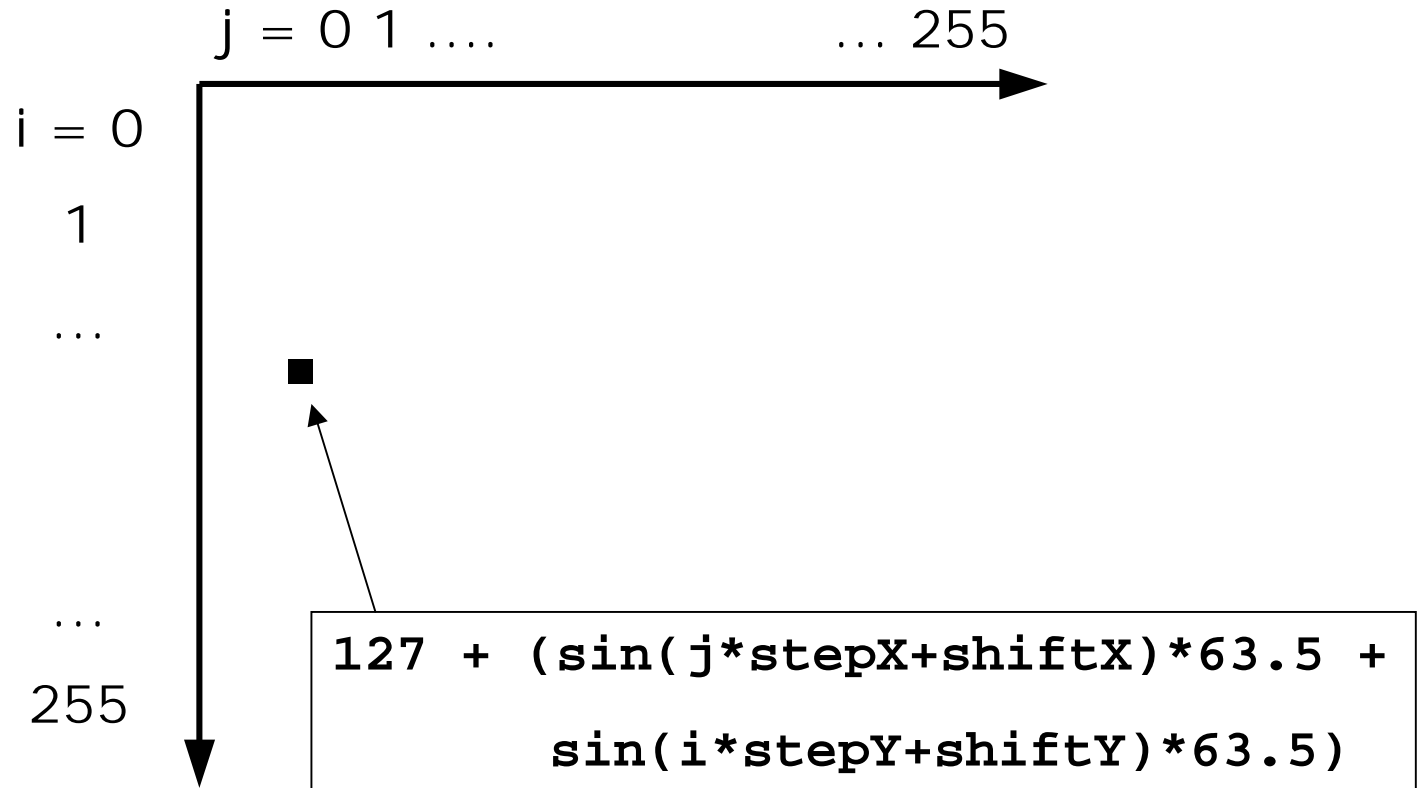
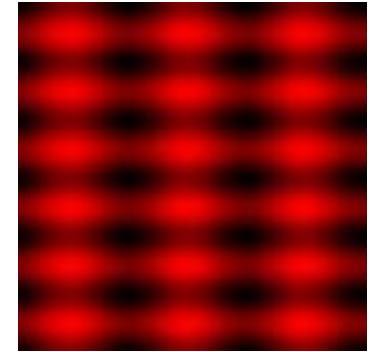
Wir brauchen:

- Schwellwert um zu ermitteln, ob ein Pixel gesetzt wird oder nicht

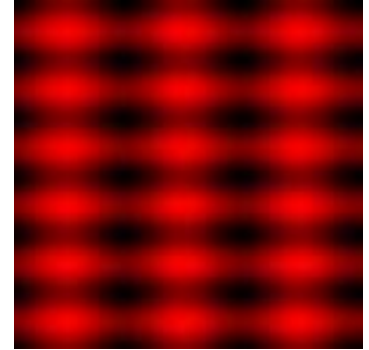
```
if (myRand() < THRESHOLD)  
    dest = myRand() * 255.0f;
```

# #2 - Sinusplasma

Idee: Überlagerte Sinuswellen in X- und Y-Richtung



# #2 - Sinusplasma



Wir brauchen:

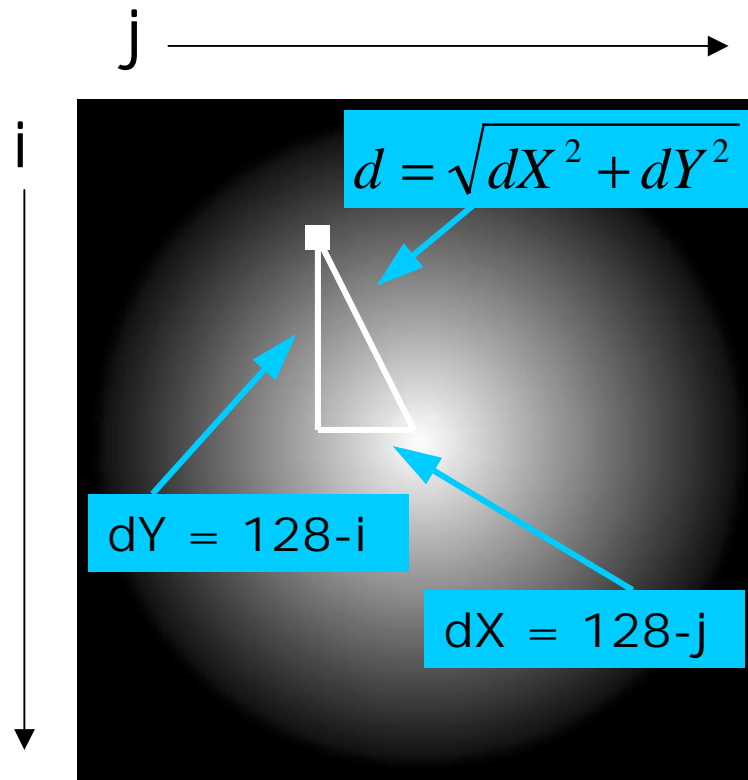
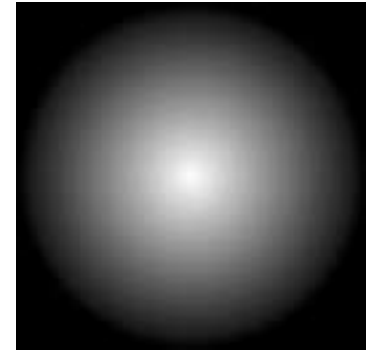
- # der Wellen in x/y Richtung  
(numX, numY)
- horizontale und vertikale Verschiebung  
(sX, sY [0..255])

```
stepX = (numX*2*M_PI)/256.0f;
```

```
shiftX = sX*(2*M_PI/256.0f);
```

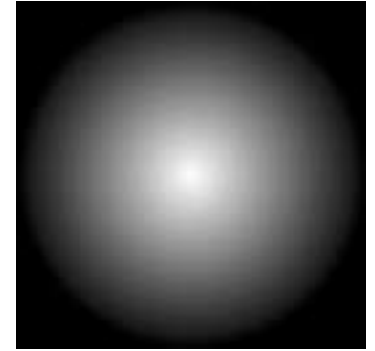
# #3 – Environment Map

Idee: Abstandsfunktion



$$c = 255 - d \text{ (gegen } < 0 \text{ prüfen! )}$$

# #3 – Environment Map



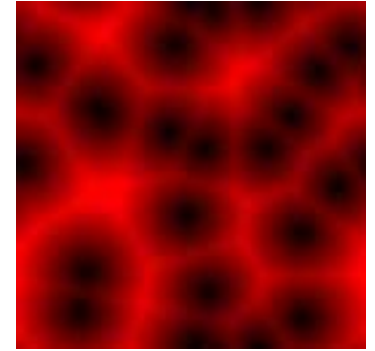
## Details

Die „Stärke“ (also die Größe) der Environment Map kann über einen Parameter zwischen 0..255 eingestellt werden. Dieser Faktor sollte zwischen 2.0 und n liegen, wobei n eine beliebige obere Grenze ist.

```
d = factor * sqrt(dy*dy + dx*dx);
```

# #4 – Zellen

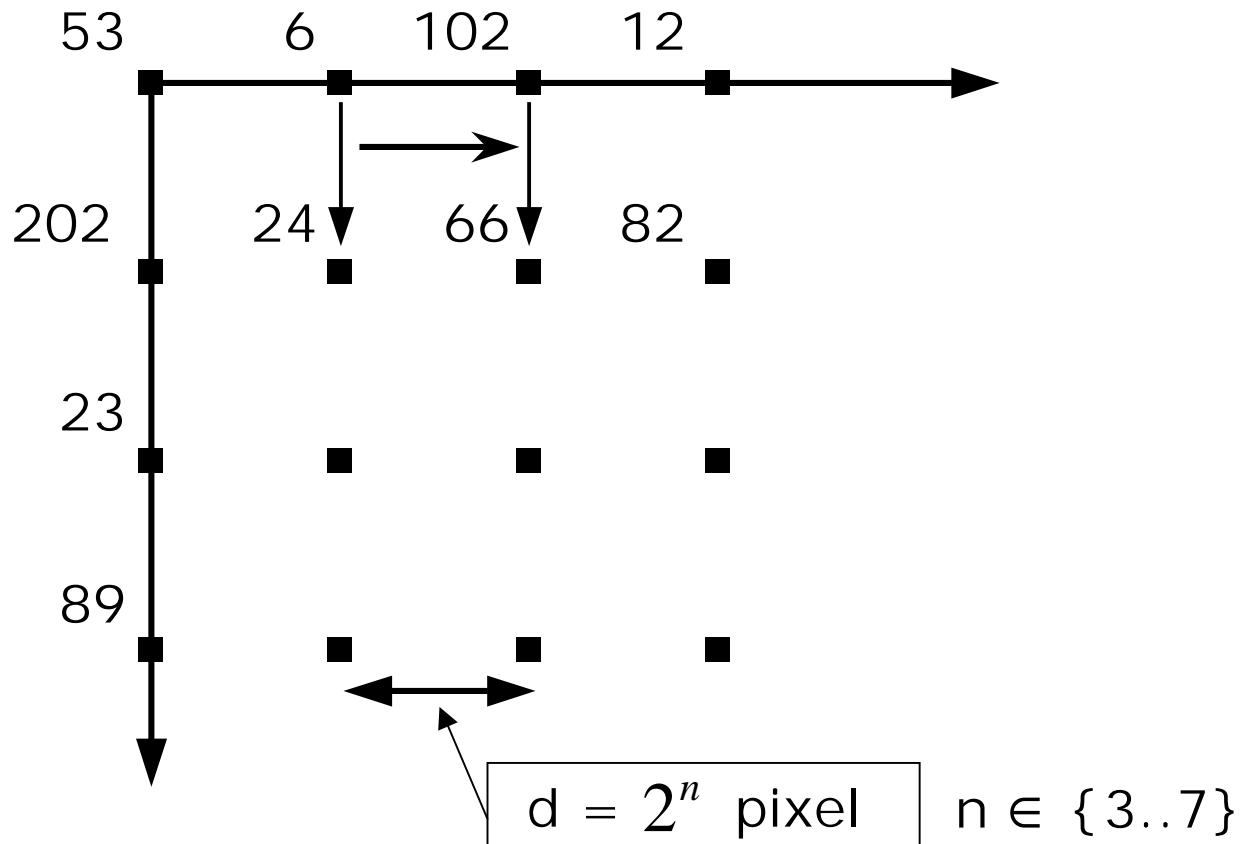
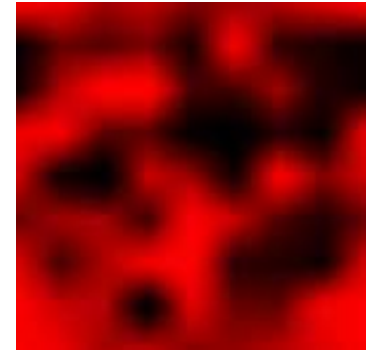
Idee: Distanzfunktion



- n zufällig verteilte Punkte generieren
- für jeden Pixel den minimalen Abstand über alle Punkte berechnen (Pythagoras)
- Farbe ergibt sich zu **(distance \* scale)**
- je größer **scale**, desto kleiner die Zellen
- leider langsam -> grafischer Ansatz möglich

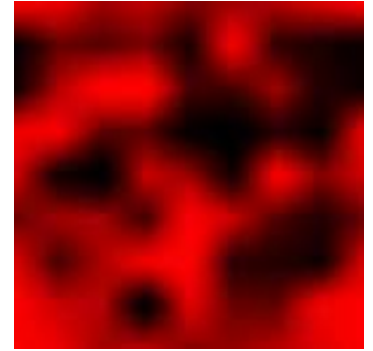
# #5 – Subplasma

Idee: 2D Interpolation



# #5 – Subplasma

## Interpolationsmethoden



- *Lineare Interpolation*

Sehr einfach, sehr schnell; eher hässlich

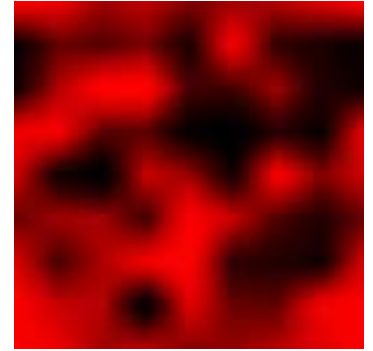
- *Kubische Interpolation (Spline)*

Sehr glatt; recht kompliziert, langsam

- *Sin/Cos Interpolation*

Gutes Ergebnis; immer noch recht kompliziert, schnell

# #5 – Subplasma



Details

Cosinus-Tabelle:

```
for (i = 0 ; i < 256 ; i++)  
    costable[i] = int(127+(cos(i*2π/256)*127.0f));
```

Gitterwerte:

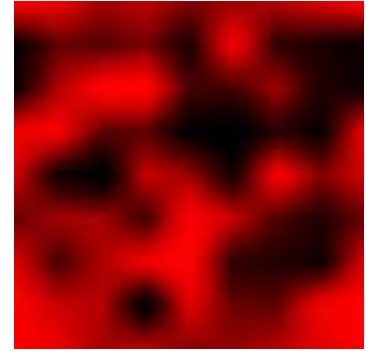
```
for (i = 0 ; i < 4096 ; i++)  
    points[i] = fMyRand(1.0f);
```

Interpolation-Innerloop:

```
tex[pos] =  
    costable[int(127+(sx+(x*stepx))*127)%256];
```

# #5 – Subplasma

Modifikationen



Für etwas andere Resultate mit *sin* interpolieren:

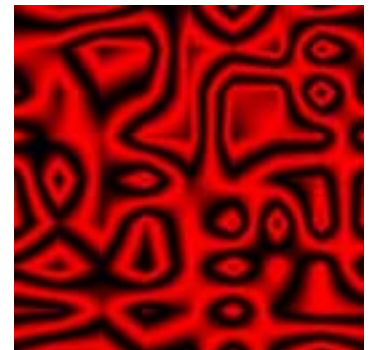
```
tex[pos] =  
    costable[int(127+sin(sx+(x*stepx)))*127)%256];
```

Gitterwerte:

```
for (i = 0; i < 256; i++)  
    points[i] = fMyRand(1.0f)*scale;
```

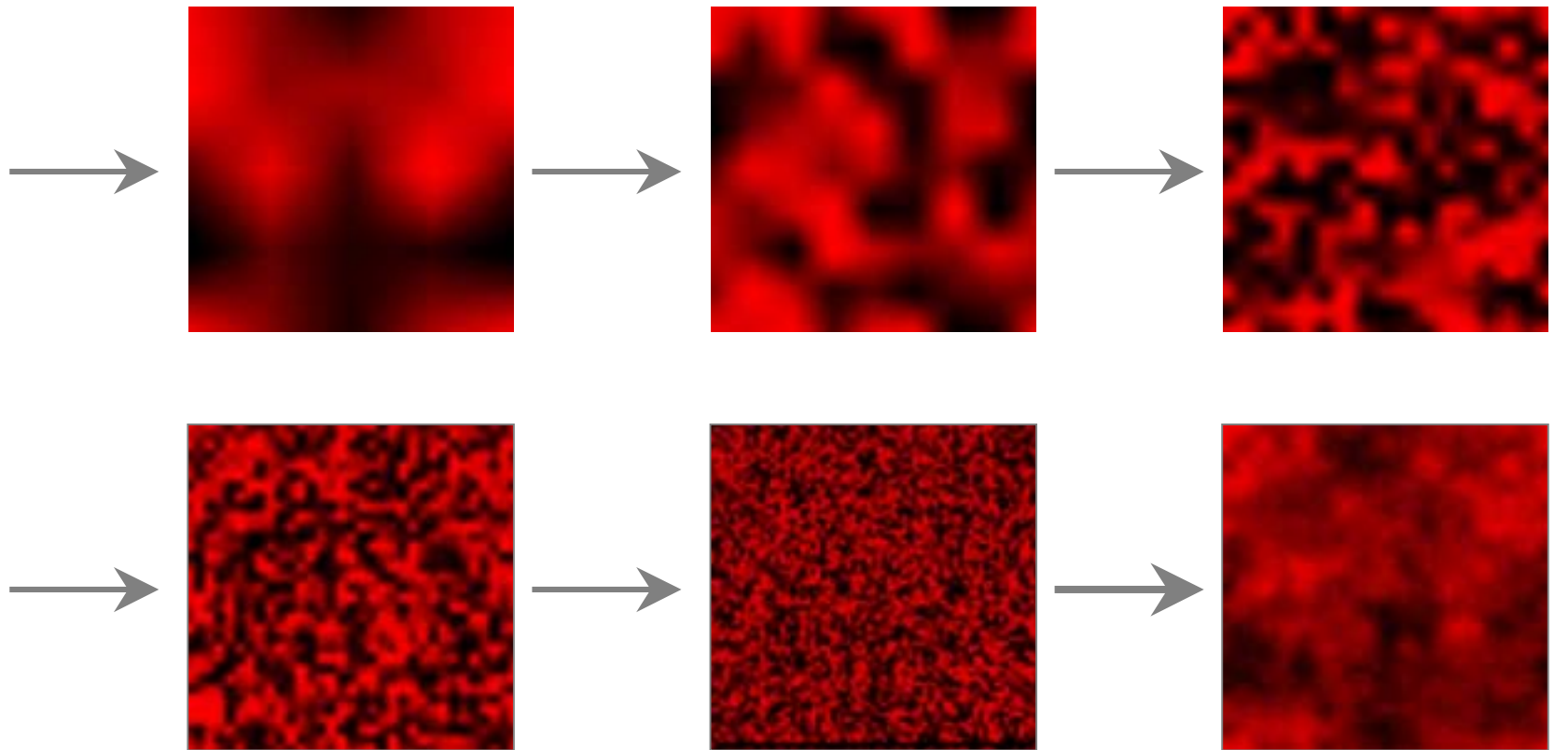
Skalierung kann beliebig sein,

also 1.0f,  $\pi$ ,  $2\pi$ , 5, 23, ...

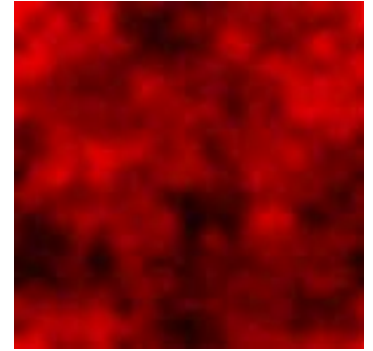


# #6 – Perlin Noise

Idee: Subplasmen addieren



# #6 – Perlin Noise

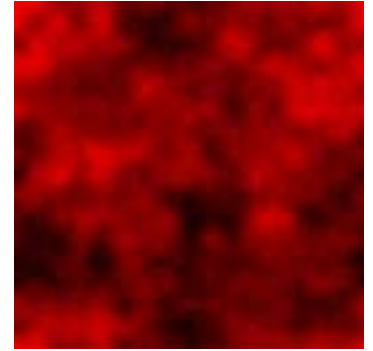


## Details

- Unterteilungen von 64..4 verwenden (5 Suplasmen)
- Mit unterschiedlicher Gewichtung (Alpha-Wert) addieren
  - je kleiner die Unterteilung, desto kleiner die Gewichtung
- Änderungen der Gewichtung können das Ergebnis stark variieren -> experimentieren (Gewichtung parametrisch ?)

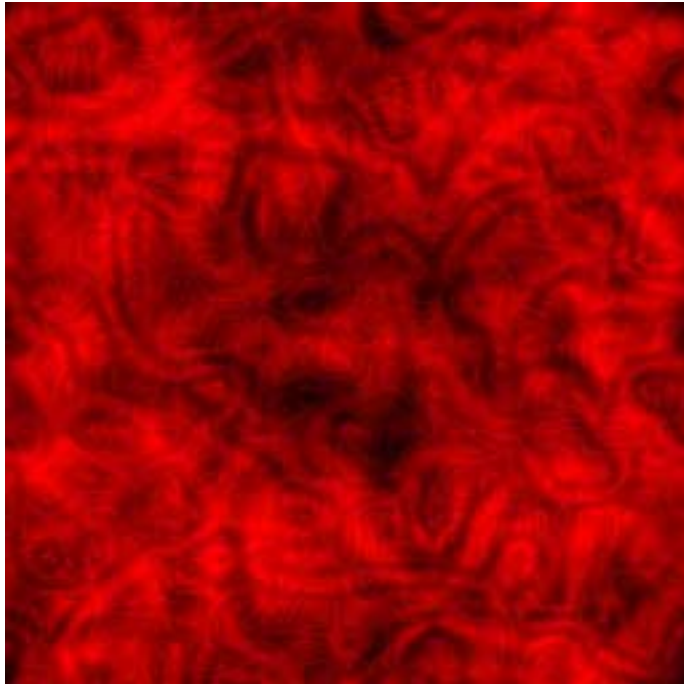
# #6 – Perlin Noise

Modifikationen

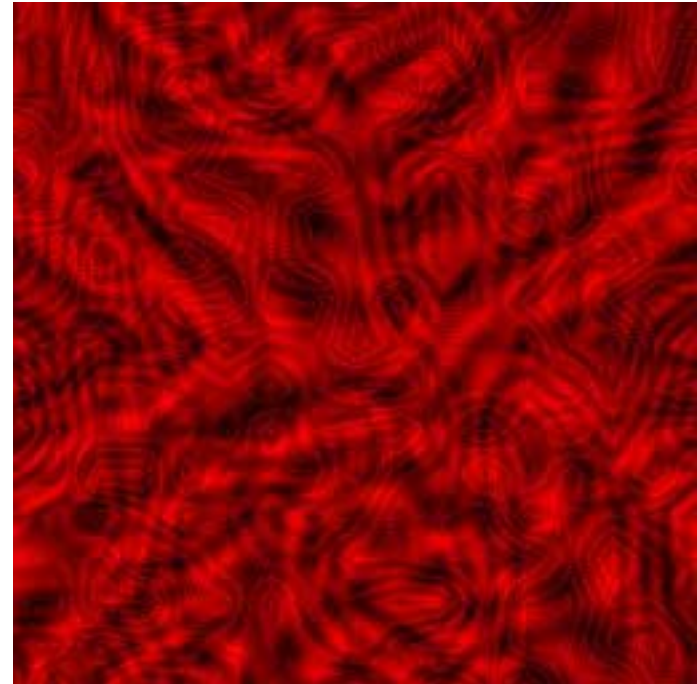


- Subplasma Skalierungen

$2 * \pi$



$4 * \pi$



# Filter

- Operationen auf einzelnen Layern
- Kombinationen von Layern
- Layer als Eingabedaten verwenden
- Operationen arbeiten entweder auf allen oder ausgewählten Farbebenen

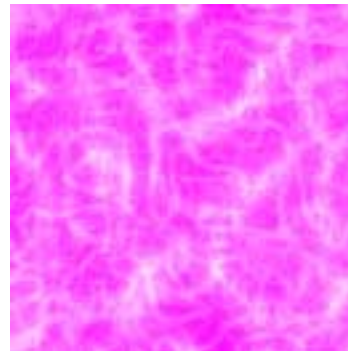
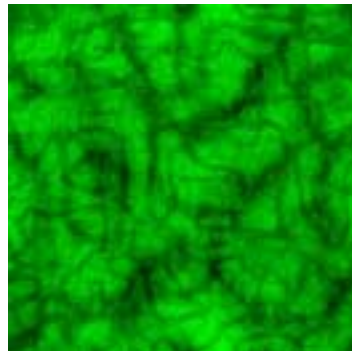
# Einfache Filter

- Layer invertieren

**dest = 255 - src;**

- Optional:

Einzelne Farbebenen invertieren



# Einfache Filter

- Layer addieren

**dest = src1 + src2;**

**dest** auf **0..255** begrenzen

- Layer multiplizieren

**dest = (src1 \* src2) / 255;**

Keine *clamp*-Operation nötig.

# Einfache Filter

- Layer mischen (*Alpha blending*)

```
dest = (src1 * a1) / 255 +  
       (src2 * a2) / 255;
```

Keine *clamp*-Operation nötig.

```
a1 + a2 = 255 = balance + a2
```

```
a2 = 255 - balance
```

# Einfache Filter

- Layer vertauschen

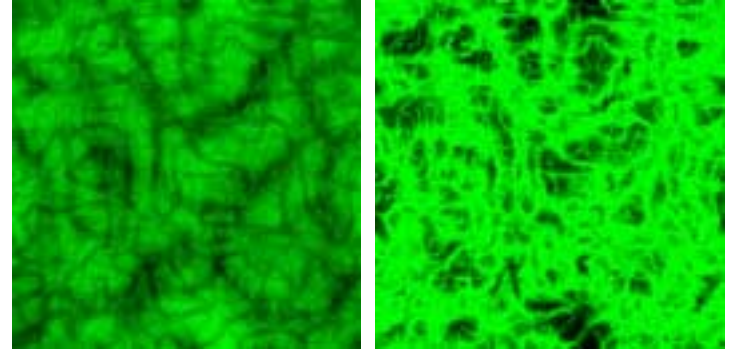
Entweder komplette Layer oder  
einzelne Farbebenen vertauschen:

```
Copy (temp, tex->layers[a]);
```

```
Copy (tex->layers[a], tex->layers[b]);
```

```
Copy (tex->layers[b], temp);
```

# Einfache Filter



- Sinuscolor

`sin()` eines Farbwertes verwenden

```
c = int(127 +  
        (127 * sin(nSines*src*step)));
```

**nSines**      Anzahl der Sinuswellen

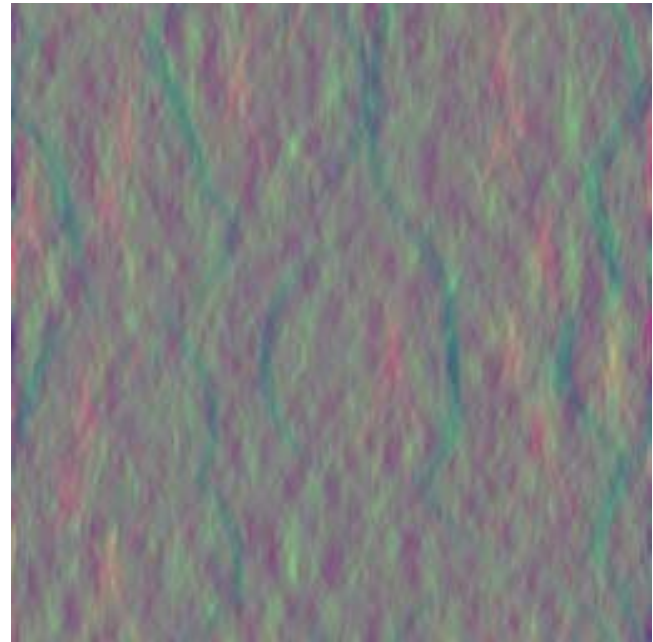
**src**            Eingabepixel

**step**            Faktor von  $\pi$ , z.B.  $\pi/128$

# Komplexere Filter

- Emboss

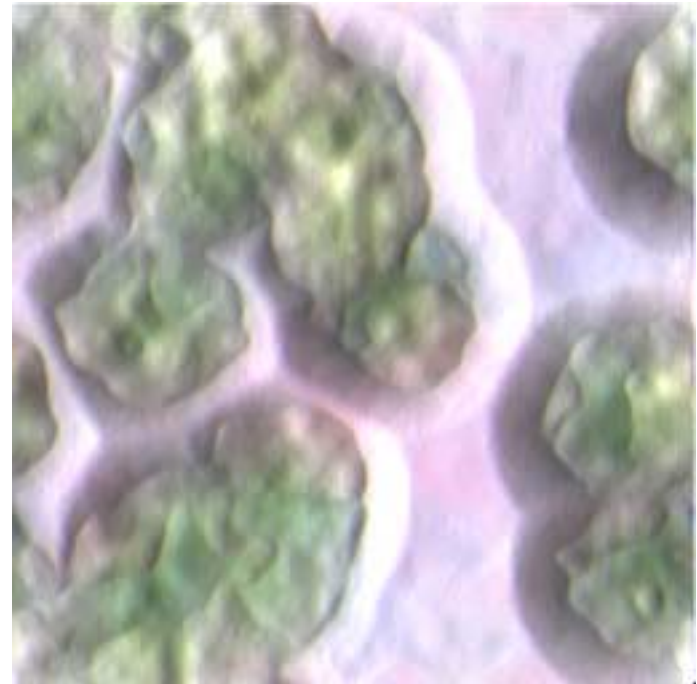
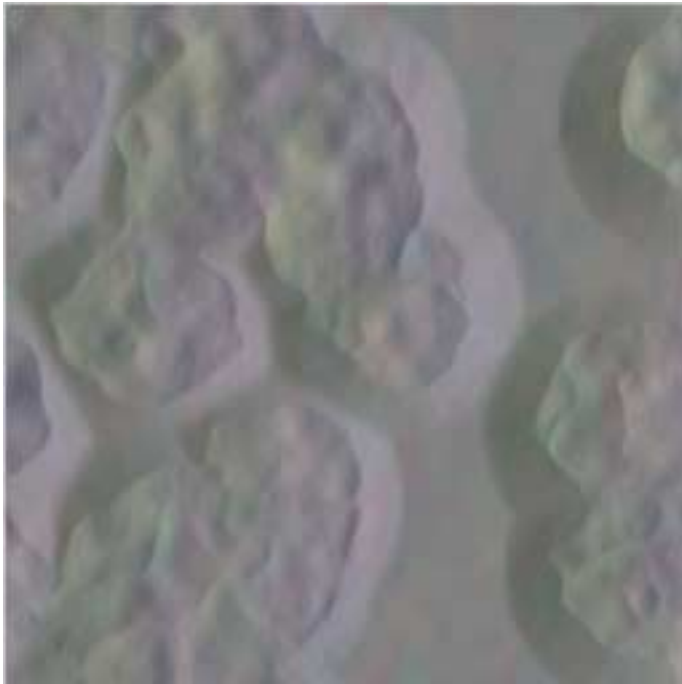
-1	0	1
-1	0	1
-1	0	1



→ Diese 6 Pixel + 128 addieren  
(für jede Farbebene wiederholen)

# Komplexere Filter

- Mit Farbebene schattieren



# Komplexere Filter

- Mit Farbebene schattieren

```
byte s;
```

```
int c;
```

```
...
```

```
s = (shadelayer->data[pos]);
```

```
c = (layer->data[pos_red] * s) / 128;
```

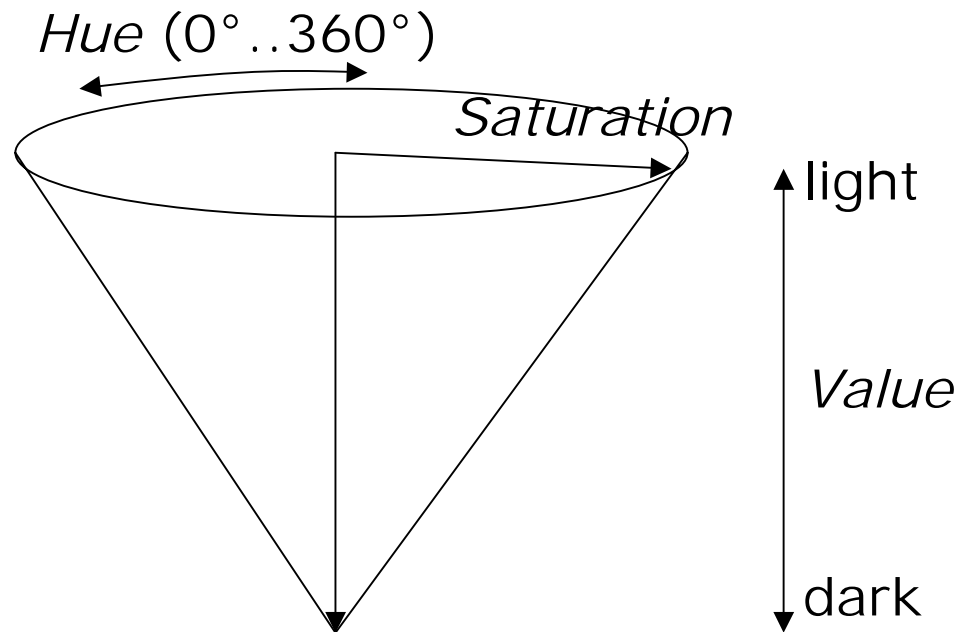
c auf 0..255 begrenzen

```
layer->data[pos_red] = c;
```

→ Das selbe für `pos_green` und `pos_blue`

# Komplexere Filter

- Farbraumtransformation (HSV)



# Komplexere Filter

- Farbraumtransformation (HSV)

```
new_hue = hue*(360.0f/256.0f);
```

```
new_sat = sat/255.0f;
```

```
new_val = val/255.0f;
```

```
...
```

```
rgb2hsv (pix_r, pix_g, pix_b, h, s, v);
```

```
h += new_hue;
```

```
if (h > 360.0f) h -= 360.0f;
```

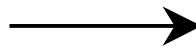
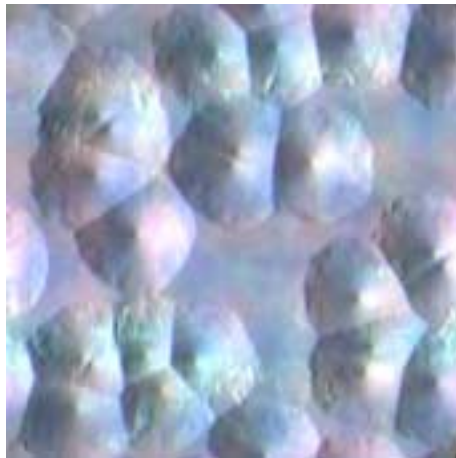
```
s *= new_sat;
```

```
v *= new_val;
```

```
hsv2rgb (h, s, v, pix_r, pix_g, pix_b);
```

# Komplexere Filter

- Farbraumtransformation (HSV)
- Wozu ?
  - ➔ Finetuning von Farben

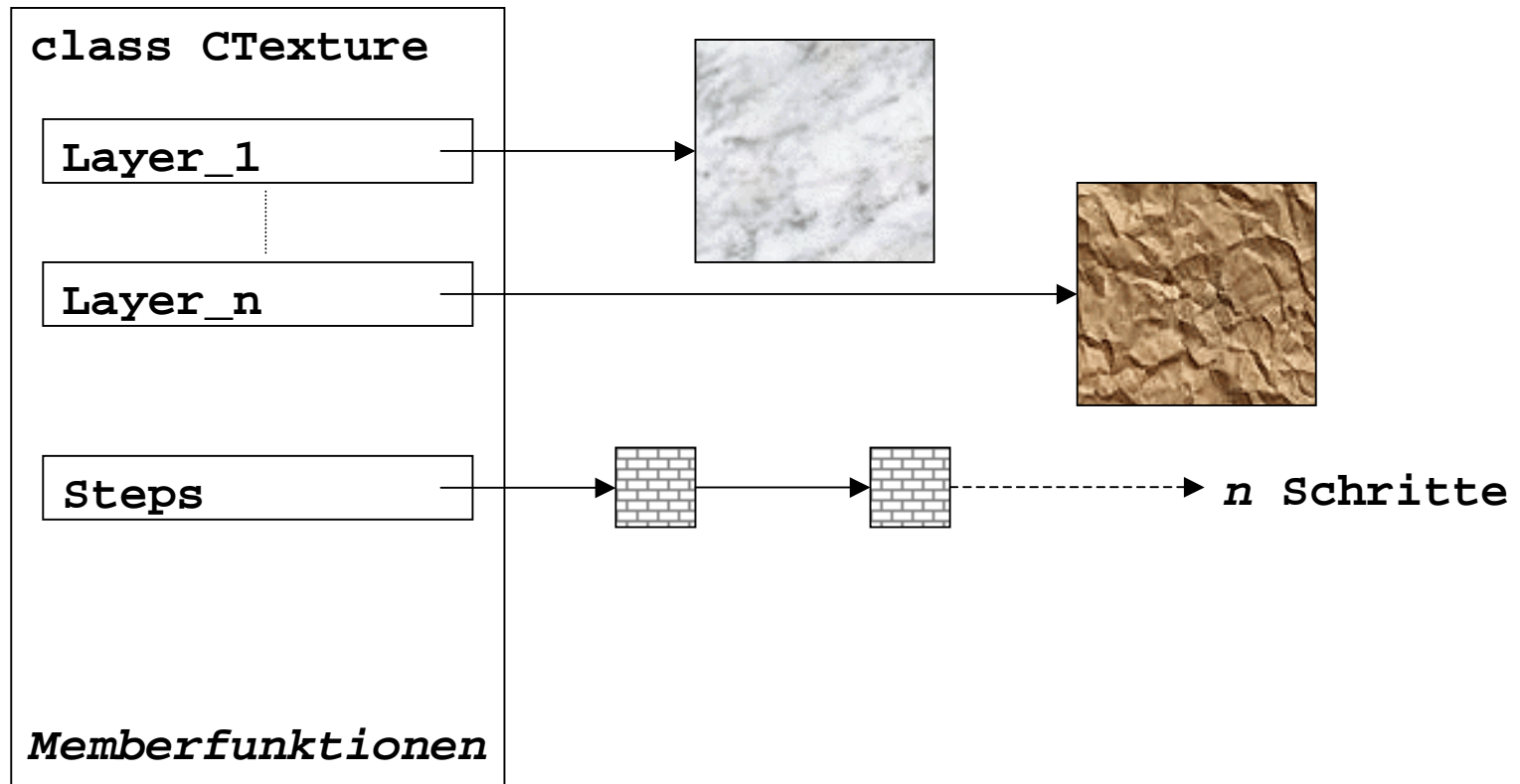


# Andere Filter

- Deformation / Morphing  
(Sinusverzerrung, Twirl, ...)
- Quellcode anschauen oder  
eigene erfinden :-)

# Implementierung

- Datenstrukturen



# Implementierung

- Texturschritte

```
class CTextureSteps
```

```
{
```

```
public:
```

```
    CStepNode *steps, *last;
```

```
    int nSteps;
```

```
    void AddStep (byte fNr, byte p[9]);
```

```
    void RemoveStep (void);
```

```
    void KillList (void);
```

```
    void WriteHTF (char *fName);
```

```
};
```

# Implementierung

- Texturschritte

```
class CStepNode
{
public:
    byte FilterId;
    byte params[9];
    class CStepNode *prev, *next;
};
```

# Implementierung

- Texturschritte / Speicherung
  - Jeden Schritt an das Ende der Liste hängen
  - für ein *undo* den letzten Schritt löschen
  - *redo* kann mit etwas Mehraufwand ebenfalls implementiert werden
  - Datei-„Format“:

1 BYTE	nSteps*10 BYTE
nSteps	Funktion-ID + Parameter

# Implementierung

- Laden
  - Entweder alle Schritte laden und einmal die Liste abarbeiten oder jeden geladenen Schritt direkt ausführen

# Implementierung

- Editor
  - Unbedingt einen Editor mit GUI schreiben!
  - Gebräuchlich: Ein Layerbereich mit 4+ sichtbaren Layern sowie ein Operationsbereich wo die Filter ausgewählt werden und die Parameter (z.B. über Radiobuttons und Slider) modifiziert werden
  - 2+ zusätzliche (unsichtbare) Layer sind normalerweise nötig
  - außerdem werden Hilfsfunktionen zum Kopieren etc. benötigt

# Weiterführendes

- Datenrepräsentation
  - Dynamischer Stack bietet mehr Flexibilität
  - Aufbau neuer Texturen auf Teilen anderer Texturen vereinfacht den Erstellprozess und bietet ebenfalls mehr Möglichkeiten

# Weiterführendes

- Pixel Shader ?
  - Definitiv möglich, z.B. *Perlin Noise*
  - Nachteil: hardwareabhängig

# Fragen, Kommentare ?

- Your turn

## Diverses

<http://w3studi.informatik.uni-stuttgart.de/~schillas/>  
andi.schilling@gmx.de