Prozedurale Texturen

>>Was nicht passt wird passend gemacht...<<

Von Carsten 'Jazzoid' Przyluczky

Inhalt

1.Einleitung

Was sind prozedurale Texturen ?

- Entstehen aus dem „Nichts“
- Inhalt bekommt erst zur Laufzeit finale Form
- Aus verschiedenen Operationen / Prozeduren zusammen gesetzt
- Man speichert nicht „Was“ sondern „Wie“

Motivation

- Spiele werden immer komplexer
- Spiele werden größer
- Flexiblitätsanforderungen steigen

Motivation

- Beispiel „Spore“
- Nahezu unendliche Welten
- Charaktere werden vom Benutzer zusammen gestellt mit schier unbegrenzten Möglichkeiten
- Mit statischem Inhalt nicht mit heutigen Rechnerkapazitäten machbar !

Motivation

- Daher prozedurale Texturen
- Lassen sich weitestgehend in Echtzeit anpassen / editieren
- Verbrauchen im Vergleich einen Bruchteil an Speicher
- Ideal für Handheld-Konsolen oder Handys

Prozedural VS Statisch

176 Bytes

18.158Bytes(jpg)

Wo ist der Haken ?

- Prozedurale Texturen sind nicht umsonst !
- Brauchen „Engine“ um erstellt zu werden, diese kann bei ungeschickter Implementierung sehr groß werden
- Nicht unbegrenzt sinnvoll (später mehr)
- Um effizient arbeiten zu können benötigt man ein spezielles Tool !

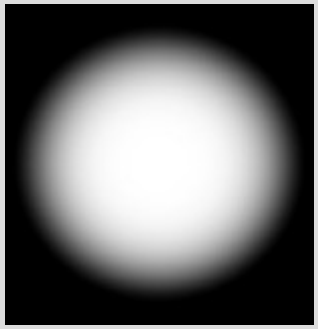
2.Beschreibung einzelner Verfahren

Voraussetzungen

- Verwendung der Hardware , insbesondere Shader 2.0
- 2 Arten von Operationen :
    - Generatoren
    - Filter

2.1 Generatoren

Kreis



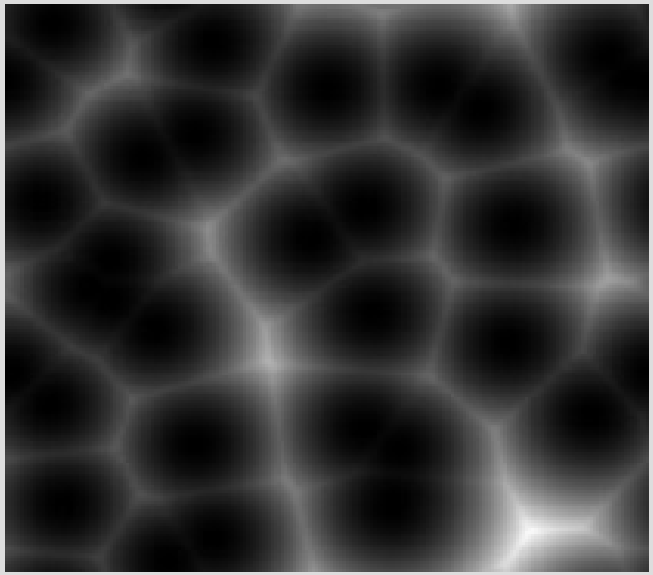
- Kreis ?! Klar, kein Thema , oder doch ?
- Problem : mit Brute-Force ‚Textur Locken und „reinpixeln“ nicht wirklich gut. Wir hätten gern **Echtzeit**
- Sinus = teuer !
- => Shader benutzen ohne Sinus

Kreis

- Idee: Den Abstand zum Mittelpunkt in Farbwert umrechnen
- Sprich : 1 – dot(Tex,Tex)
- Skalarprodukt eines Vektors mit sich selbst ergibt Betrag zum Quadrat
- Farbverlauf beeinflussen :
  1- pow(dot(Tex,Tex),smoothness)

Tex steht hier für die Texturkoordinaten

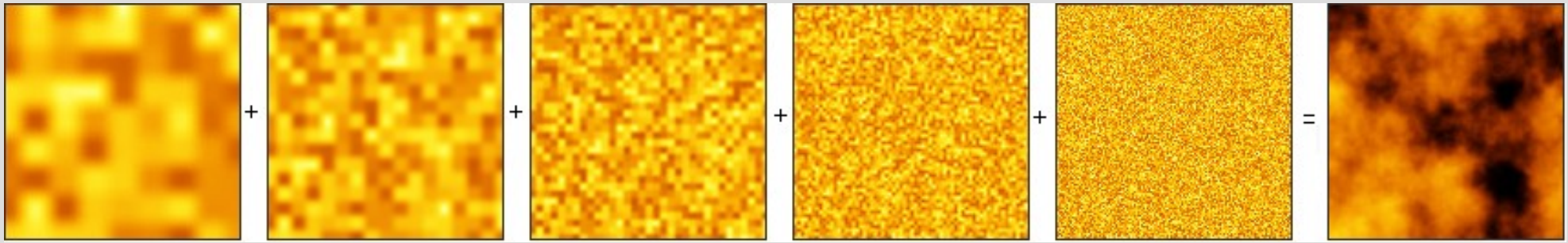
Cells



- Idee : Eine zufällige Punktmenge M auswürfeln
- $\forall$ Punkte $\notin$ M rechne Abstand zum nächsten Punkt aus M in Farbwert um
- Zur Beschleunigung wird oft eine Art BSP benutzt.

Perlin Noise

- Gut für Wolken , Heightmaps
- Idee : Überlagerung der selben Funktion mit verschiedenen Frequenzen
- Starten mit niedrigen Frequenzen , dann immer höhere , zwischen Werte werden interpoliert (Weichzeichnen)
- Die so entstehenden „Layer“ werden kombiniert

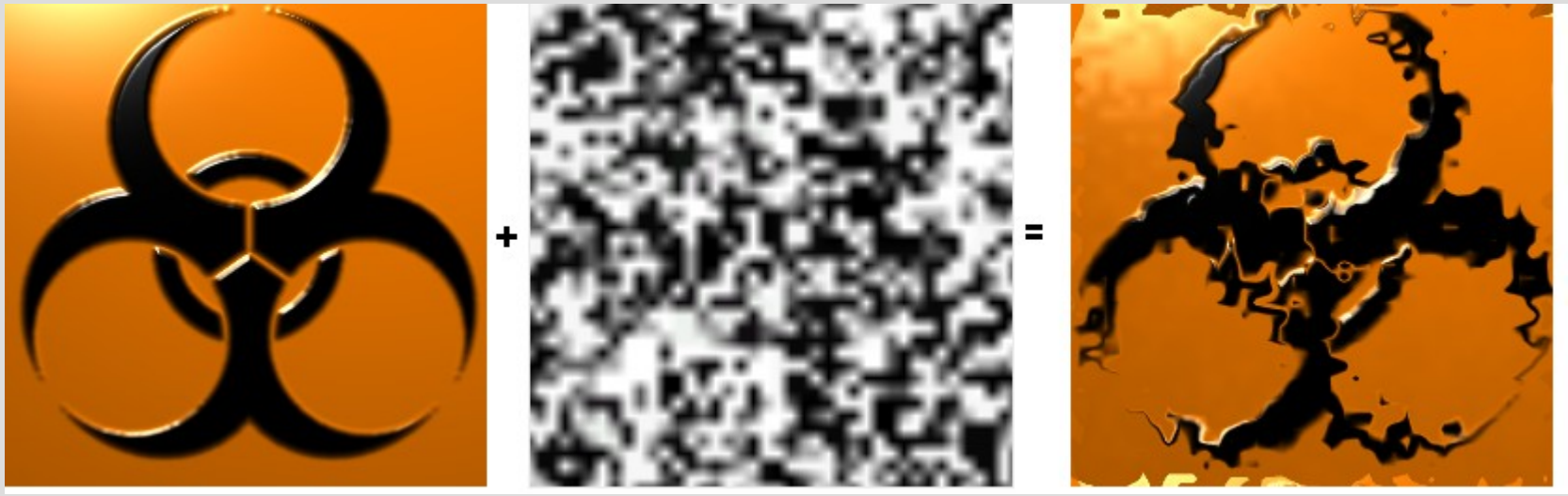


Freq.	1	2	4	8	16
Ampl.	1	½	¼	1/8	1/16

[1]

2.2 Filter

Distortion



- Idee : Textur als „Verzerrvorschrift“ nehmen
  Blauanteil der Pixel bestimmt Verschiebung in X, Grünanteil in Y Richtung

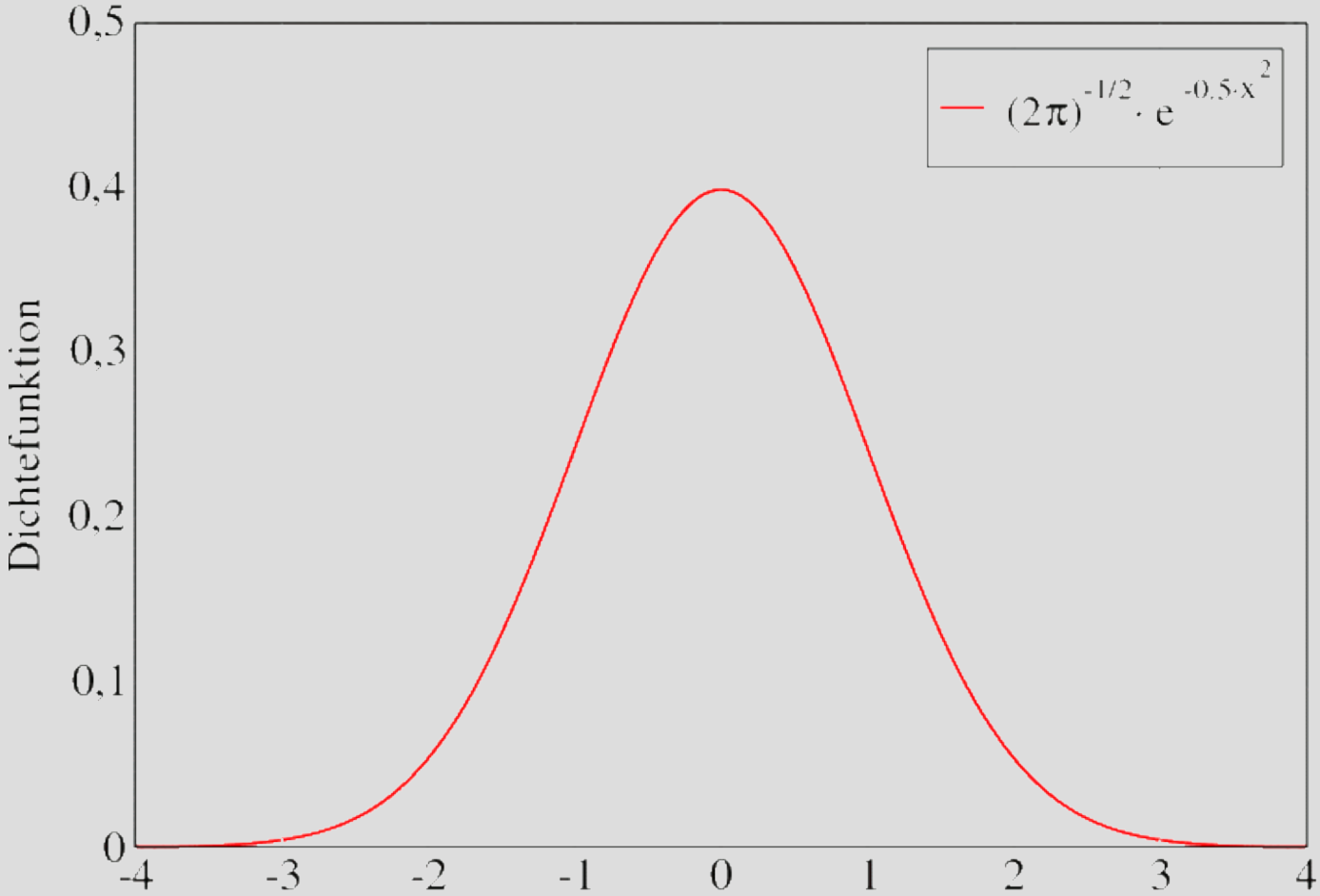
Twirl

- Idee : Textur Koordinaten in Polarkoordinaten umrechnen (x/y nach Winkel/Radius)
- Dann in Abhängigkeit zum Radius , einen Wert zum Winkel addieren
- Und zurückrechnen

Gaussian Blur

- Idee : Farbe eines Pixel wird aus den Umliegenden berechnet . Anteile der einzelnen Pixel durch Gewichte bestimmt

Gaussian Blur

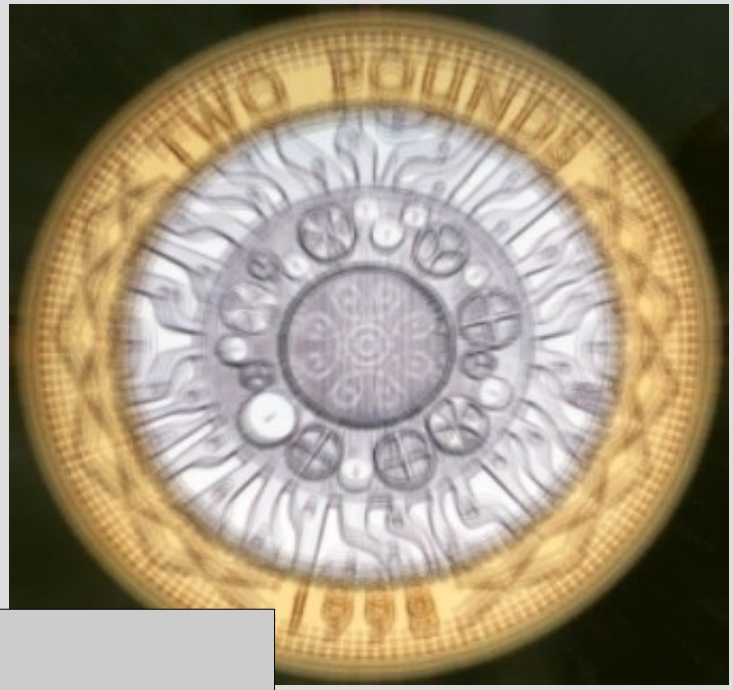


[4]

- Gewichte mittels Gauß-Kurve berechnen
  => nichtlinear verteilte Gewichte

Radial Blur

- Idee: Die Textur mit einer skalierten Version ihrer selbst addieren.



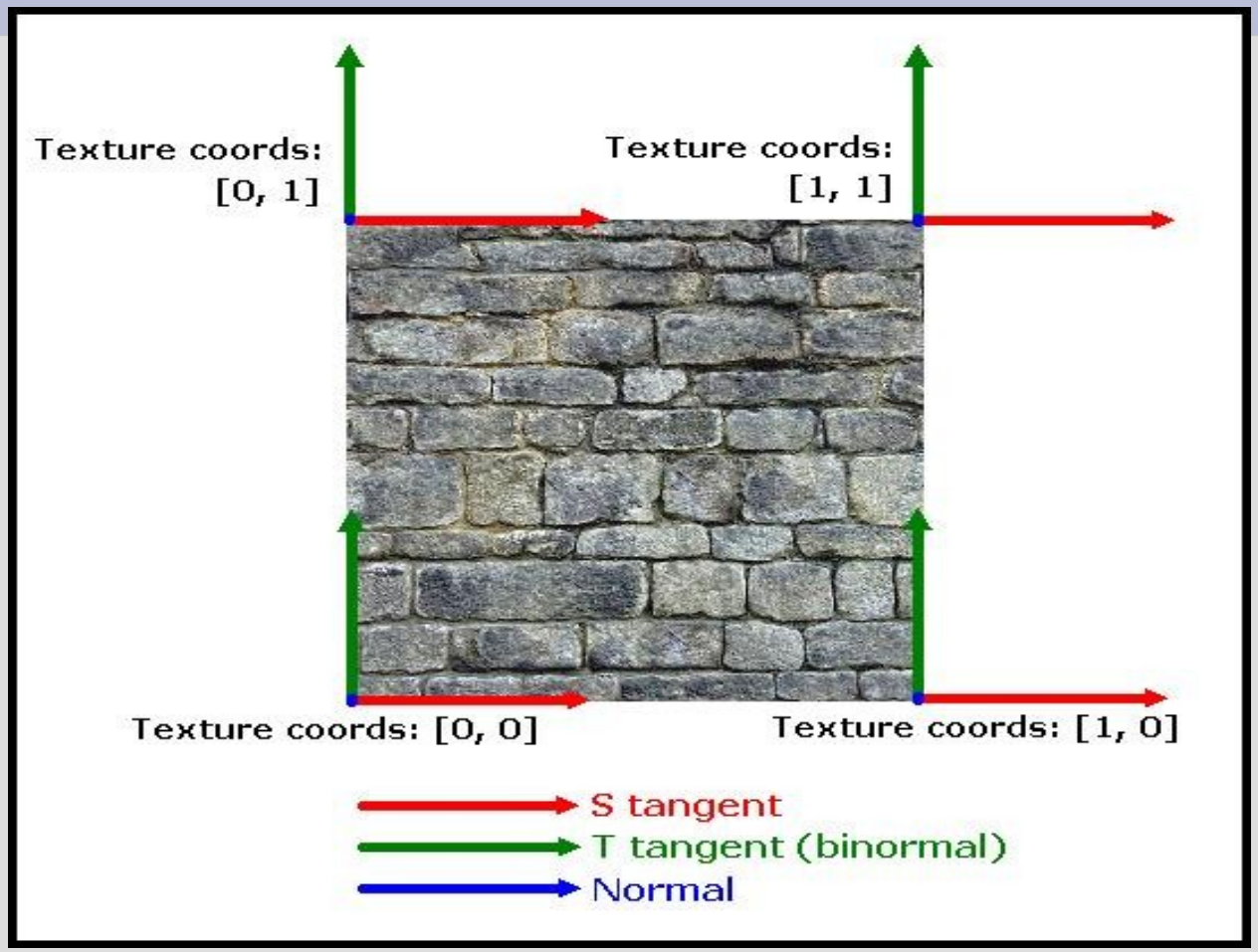
- Pseudocode:

```
for(int i = 0; i<times; i++)
{
      col = col + tex2D( BaseMap, TexCoord );
     faktor = faktor + amount;
}
```

Bump Mapping

- Beleuchten einer Textur
- Tiefeneindruck erzeugen / verstärken
- Idee : Textur erstellen, die für jeden Pixel der Orginaltextur die Normale speichert (Normal Map) R/G/B = X/Y/Z
- Beachte Farbkanal nur Werte von 0 bis 1

Bump Mapping - Begriffe



[2]

- Tangenten : Lage der Textur in X Richtung
- Binormalen : Lage der Textur in Y Richtung

Bump Mapping – Vertex Shader

- Jeder Vertex enthält neben der Normalen auch Tangente und Binormale
- Diese mit Viewmatrix multiplizieren
- View- und Lightvektor in Tangentenspace umrechnen :
  ViewDirTan.x = dot(Tangent, ViewDir)
  ViewDirTan.y = dot(Binormal, ViewDir)
  ViewDirTan.z = dot(Normal, ViewDir)
- Lightvektor analog

Bump Mapping – Pixel Shader

- 3 Komponenten bilden finale Farbe
  1. Ambientes Licht (AmbientColor*OrginalColor)
  2. Diffuses Licht (N•L)*DiffColor*OrginalColor
  3. Spiegelung der Lichtquelle $(R \bullet V)^{Power}$
- Normale aus Normal-Map holen
- Reflexionsvektor berechnen (Phong Shading)
  $R = 2(N \bullet L)N - L$
- Die 3 Komponenten addieren

Range

- Begrenzen der Farbwerte
- Für jeden Kanal :
  Wert unter Minimum ? => Wert = Minimum
  Wert über Maximum ? => Wert = Maximum
- Einfach , aber **sehr** nützlich (dazu später mehr)

3.Tipps & Tricks

Standards setzen

- Häufige Parameter standardisieren
  Beispiel : Rechtecke und Kreise meist weiß
  daher Subtyp weißes Rechteck,
  weißer Kreis
- Gewinn : Es müssen nur die Dimensionen
  gespeichert werden
- Sinnvoll : Wenn alle Texturen fertig
  => Auswertung machen!

Standards setzen

- Hier kommt der Range Operator ins Spiel
- Erst alle Rechtecke / Kreise, welche gleiche Farbe haben, sollen in weiß erzeugen
- Dann mit Range einfärben
- Gewinn : Farbe für beliebig viele gleichfarbige Primitive nur **einmal** abspeichern

Recycling (Eine für Alles)

- **Sehr wichtig !**
- Wir nutzen **eine** Funktion für :
  - Rechtecke (Textur = NULL)
  - Kopieren
  - **Alle** Shader
- Kapselnde Funktionen setzten Parameter
- Redundanten Code vermeiden
- Bonus : Code leichter zu warten

Shader „richtig“ laden

- Meist wird D3DXCompileShaderFromFile benutzt. Ok , aber geht kleiner
- Idee : Die Shader mittels VSA und PSA vor-kompilieren und diesen Binärcode nutzen
- HLSL ? Kein Problem ! Rendermonkey[3] von ATI kann HLSL in Shader Assembler umwandeln
- Nutzen : Objektdateien meist kleiner als Text

Statische Daten nutzen

- Manchmal ist der Weg mit prozeduralen Methoden teurer als die statische Variante
- Daher statische Daten nutzen und in Prozess mit einbeziehen
- Beispiel : Silhouetten in 1-Bit Textur speichern für komplexe Umrisse / Schriftarten

Verwenden was schon da ist

- Alle Kombinationsoperationen werden mit den Standard-Multi-Textur-Befehlen realisiert sprich D3DTSS_COLOROP setzten.

4.Quellen

- [1] http://www.archadegames.com/articles.php?aname=texgenpt4mod.php
- [2] http://www.blacksmith-studios.dk/projects/downloads/bumpmapping_using_cg.php
- [3] http://ati.amd.com/developer/rendermonkey/index.html
- [4] http://de.wikipedia.org/
- http://dojo.windigo-design.de
- http://www.windigo-design.de

Ende

Vielen Dank !