

# Implementierung und Evaluierung von Video Feature Tracking auf moderner Grafik Hardware

Diplomarbeit vorgelegt von  
Sebastian Heymann

Betreut von  
Prof. Dr. Bernd Fröhlich  
Bauhaus Universität Weimar

Dr. Aljoscha Smolic  
Fraunhofer Heinrich Hertz Institut  
Berlin

21. September 2005

# Übersicht

- Motivation und Zielstellung der Arbeit
- SIFT - *Scale Invariant Feature Transform*
- GPGPU - *General Purpose Computations on the GPU*
- SIFT Implementierung auf der GPU
- Zusammenfassung
- Resultate

# Motivation und Zielstellung der Arbeit

- Computer Vision:
  - Computer sollen Bilder “verstehen”
  - Extraktion von Bildinformationen
  - Interpretation von Bildinformationen
- Anwendungsbeispiele:
  - Szenenrekonstruktion
  - Orientierung in Szenen
  - Objekterkennung



# Allgemeiner Ansatz bei Computer Vision

- Finden von Punkten mit hoher Einzigartigkeit und somit hoher Wiedererkennbarkeit ( Featurepunkte )
- Häufig verwendete Kriterien:
  - Kanten
  - Ecken
  - Farben
  - Extrema
- “Lernmenge” von Punkte wird durch Eingangsbild(er) erzeugt
- In späteren Bildern wird nach Übereinstimmungen mit der Lernmenge gesucht

# Probleme

- Je höher die Qualität der Featurepunkte desto höher ist der Berechnungsaufwand
- Qualität? Wiedererkennen von Featurepunkten nach:
  - Positionswechseln
  - Rotationen
  - Blickrichtungsänderungen
  - Beleuchtungsvarianzen
- Echtzeitanwendung nur eingeschränkt möglich

Zielstellung: Implementierung eines Featureverfahrens auf CPU und GPU zum anschließenden Vergleich

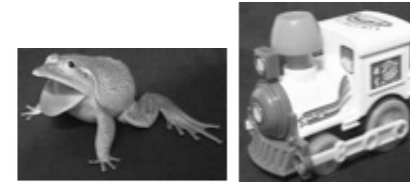
# SIFT - *Scale Invariant Feature Transform*

- David Lowe et al. 1999-2004
- Methode zur Generierung von Featurevektoren
  - Besitzt Invarianzen gegen: Rotation, Skalierung, Helligkeits/Kontraständerungen, affine Transformationen

Schritte zur Generierung von SIFT Features:

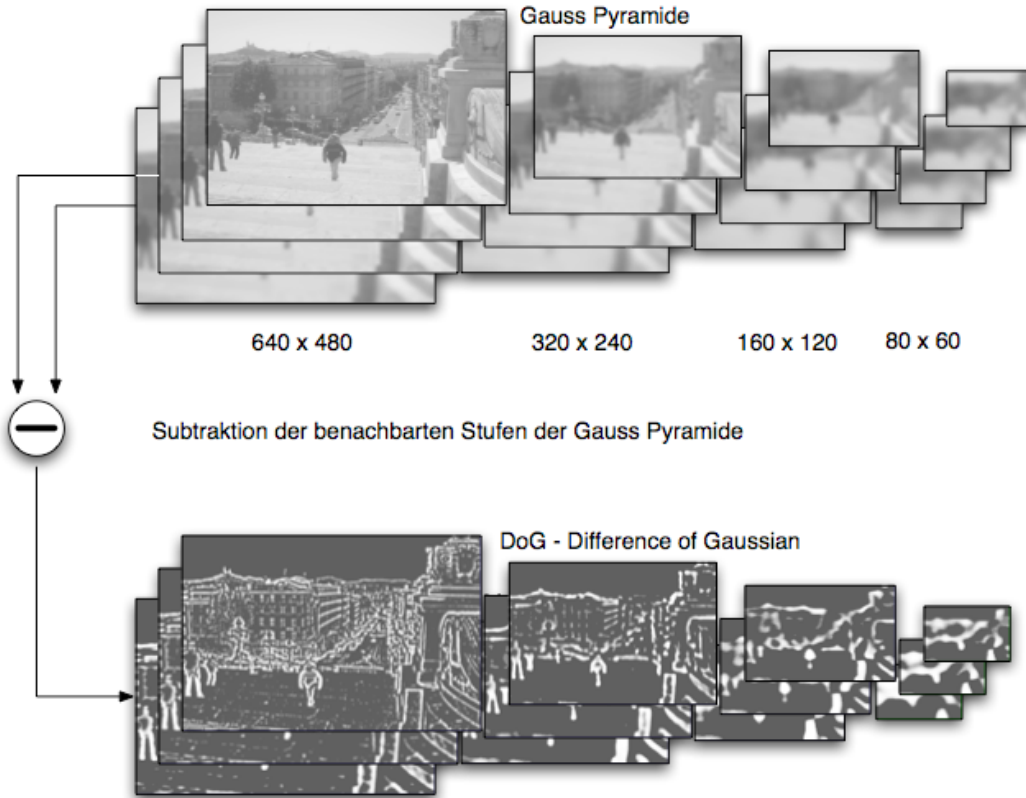
1. Suche nach globalen Extrempunkten in den Bildfrequenzen
2. Filterung und Lokalisierung der Interessenpunkte
3. Richtungsberechnung der Punkte
4. Berechnung der Featurevektoren

LoweSift03



# 1. Suche nach Extrema

## 1.1 DoG - *Difference of Gaussian* Pyramide

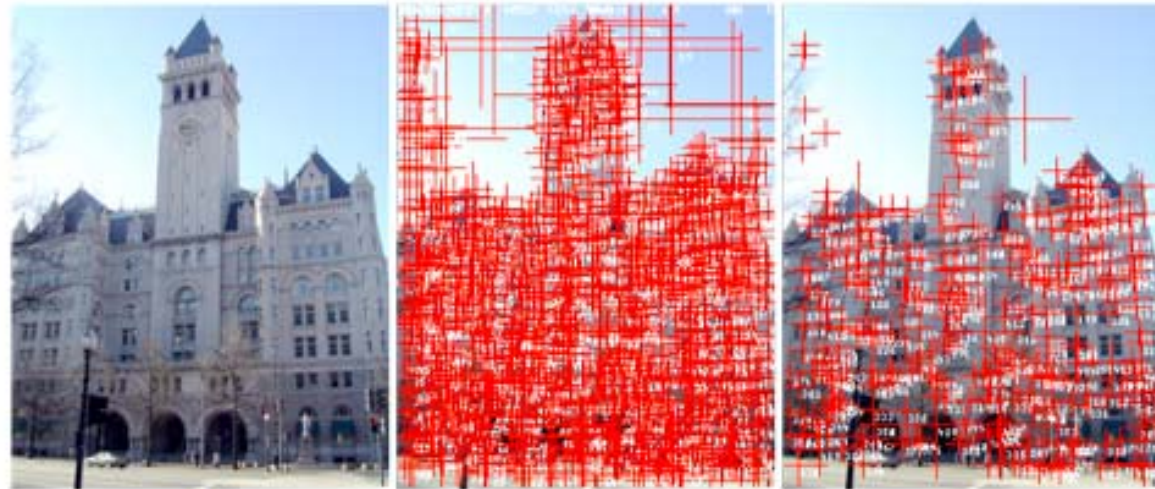
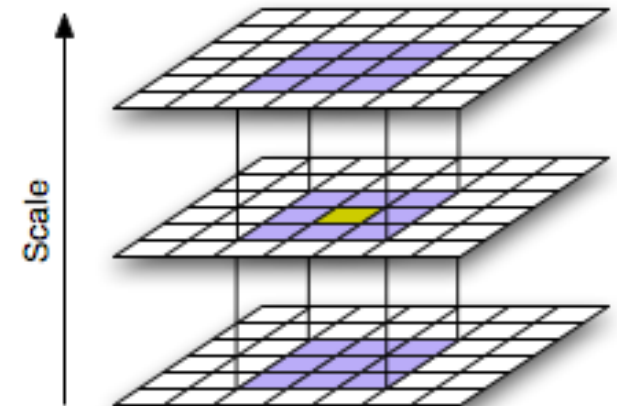


- Wiederholte Gaußfaltung des Originalbildes in verschiedenen Skalierungen
- Subtraktion der in der Pyramide benachbarten Bilder
- dadurch Trennung der Frequenzen des Eingangsbildes
- Scale Space/Skalenraum [Witkin, 83]

# 1.2 Suche nach Extrema & 2. Filterung der Extrem

## 1.2 Suche nach Extrema & 2.1 Filterung der Extrempunkte

- Schwellwertfilterung in der DoG Pyramide
- Prüfen verbleibender Punkte auf globales Minimum / Maximum
- Entfernen von auf Kanten liegenden Punkten



a) Eingangsbild

b) lokale Extrempunkte

c) Globale Extrema / nach Kantenfilterung

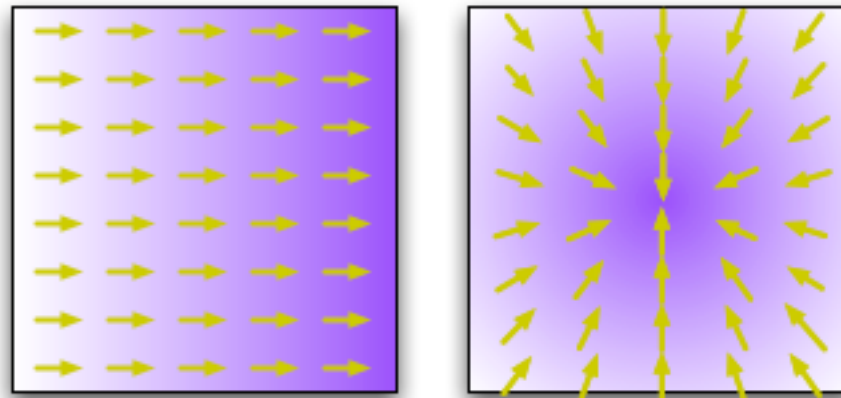
(a)

(b)

(c)

# 3. Richtungsbestimmung der verbliebenen Punkte

- Vorberechnungen der Gradientengröße  $m(x, y)$  und Richtung  $\theta(x, y)$  für die gesamten Skalenraum Pyramide
- Richtung des potentiellen Keypoints wird berechnet (Rotationsinvarianz)
- Richtung und Größe (Frequenz des Extrempunktes in der DoG Pyramide) sind Basis der nachfolgenden Berechnungen

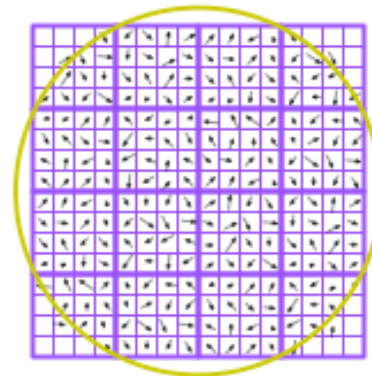
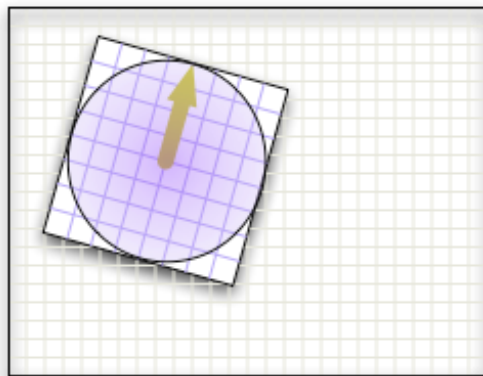


$$\theta(x, y) = \tan^{-1}(L(x, y + 1) - L(x, y - 1) / L(x + 1, y) - L(x - 1, y))$$

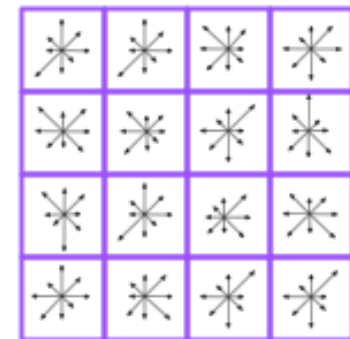
$$m(x, y) = \sqrt{(L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2}$$

# 4. Berechnung der Featurevektoren

- 4x4 Gradientenfenster pro Featurevektor
- Gradientenhistogramm aus je 4x4 Samples pro Fenster in 8 Richtungen
- Gaußgewichtung um den Mittelpunkt des Keypoints
- daraus resultiert ein 128-elementiger Feature Vektor



Bildgradienten

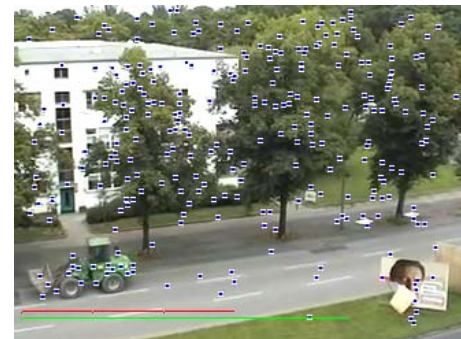
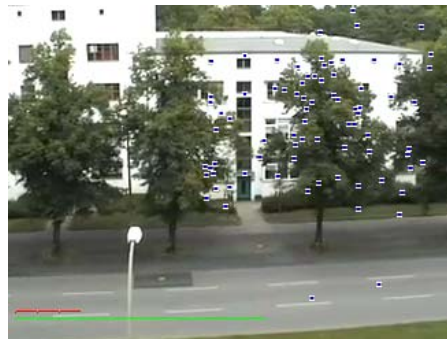


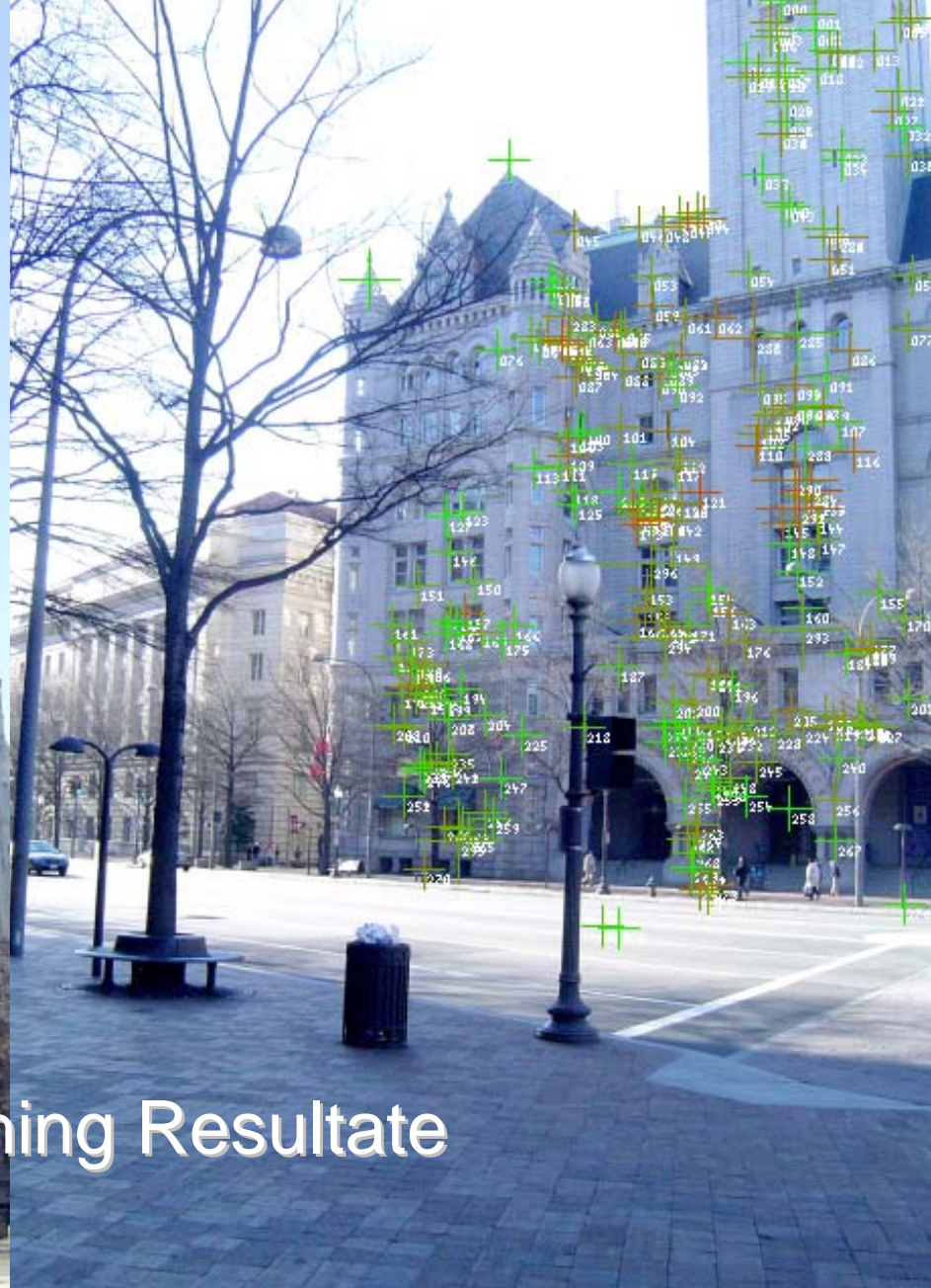
Keypoint Deskriptor

- Normalisierung des Vektors erzeugt Helligkeitsinvarianz

# Keypoint Matching

- Berechnung des euklidischen Abstands der zu vergleichenden Vektoren
- Abstand kleiner als *Schwellwert*, dann Übereinstimmung, sonst nicht.
  - Brute Force - Vergleichen jedes einzelnen Vektors
  - kd-Baum
  - BBF (Best-Bin-First)
    - Modifizierter kd-Baum
    - 95% Erfolg bei besserer Performance als Standard kd-Baum

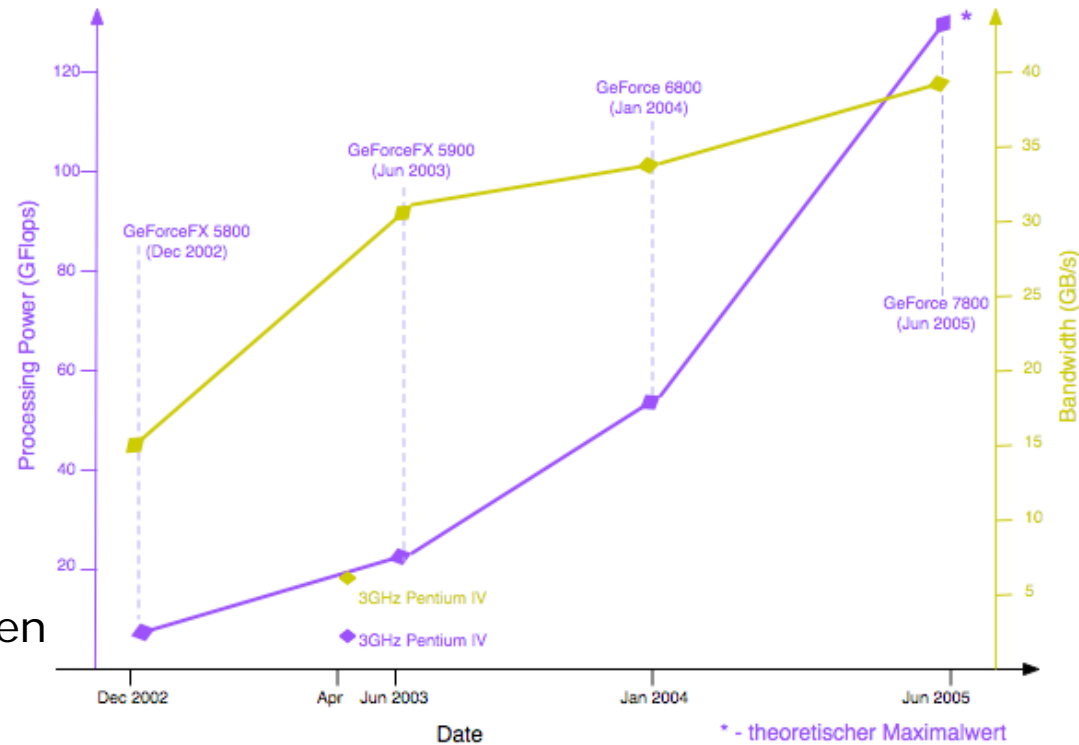




# Keypoint Matching Resultate

# Warum eine Implementierung auf Grafik Hardware?

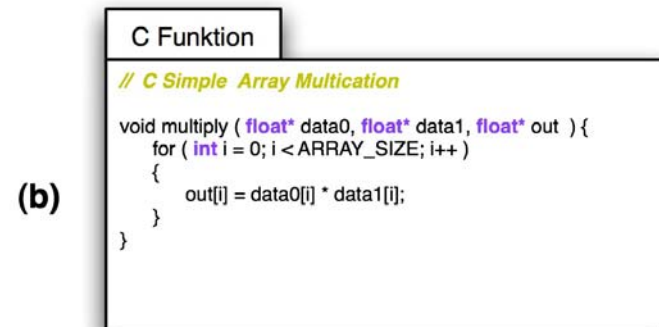
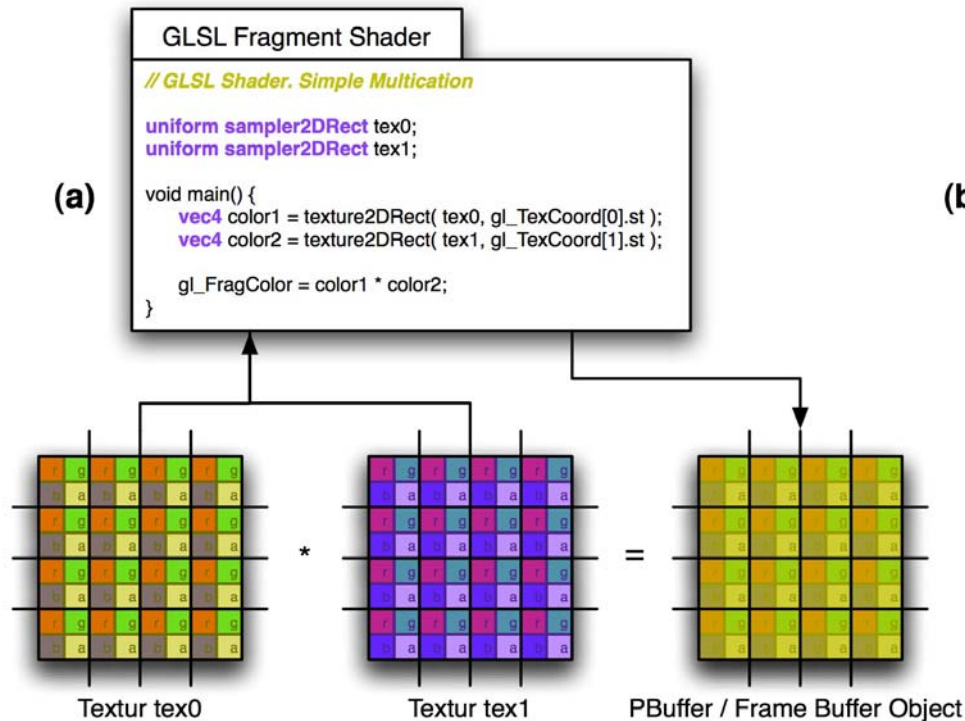
- Leistungsstarker Parallelprozessor
- Durch Videospieleindustrie  
sehr starkes Wachstum
- Beste "Rechenleistung pro Dollar"
- Spitzenwerte bei Fließkomma-  
operationen nicht auf CPUs  
erreichbar
- Nicht-bilderzeugende Berechnungen  
nach Anpassung von Algorithmen  
möglich



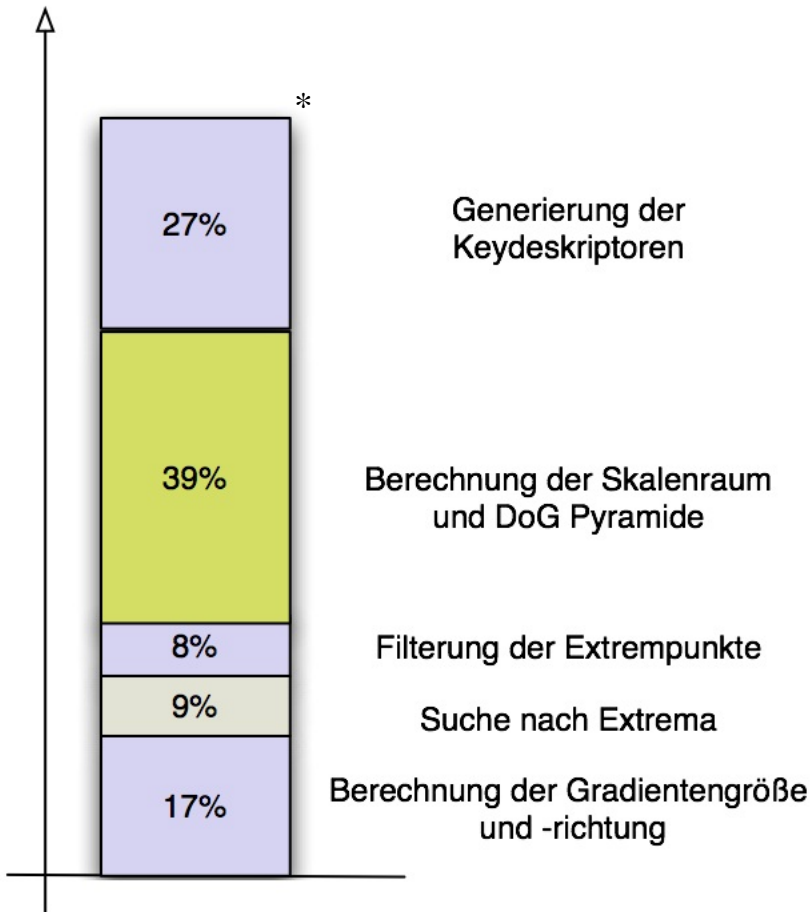
Leistungsentwicklung bei *nVidia* GPUs 2002-2005

# GPGPU - General Purpose Computations on the GPU

- Nutzen der Rendering Pipeline für allgemeine Berechnungen
- Berechnungen über Vertex- und Fragment Programme, die auf Geometrierepräsentationen und Texturdaten ausgeführt werden



# Umsetzung auf GPU



- 1:1 oder n:1 Operationen
- 1:n oder n:n Operationen  
nicht gut möglich
- Teile des SIFT Verfahrens können gut in Hardware realisiert werden, besonders:
  - Gaussfaltung/Differenz
  - Gradientenberechnungen
  - Suche nach Extrema

# Berechnung der DoG-Pyramide

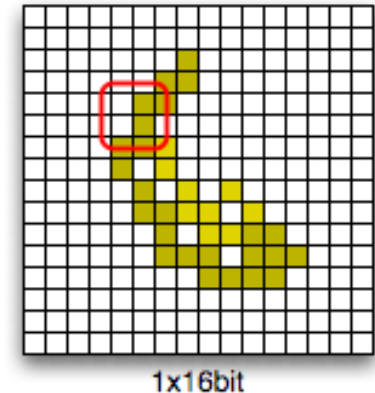
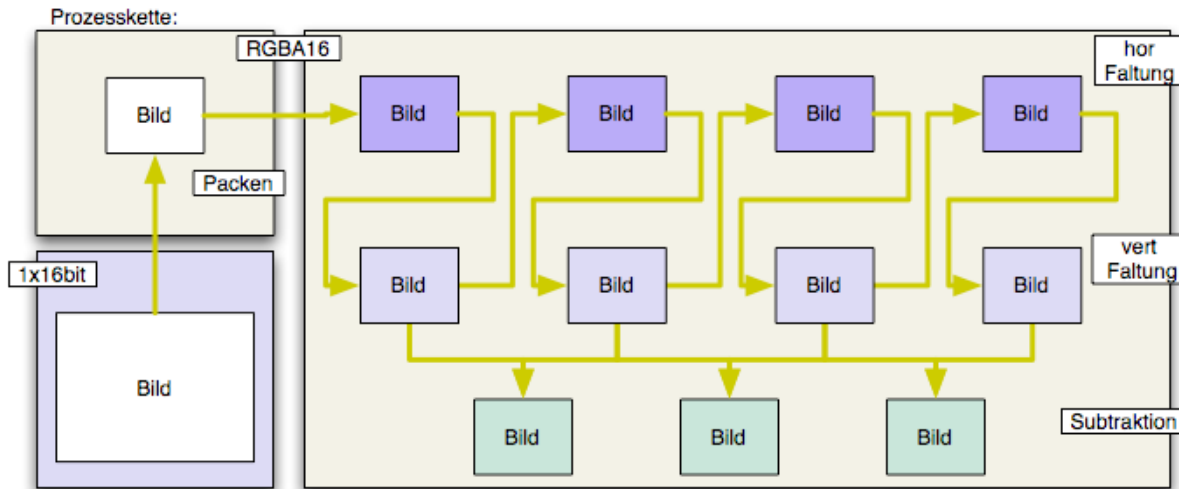
- Intuitive Übersetzung:

1. 1D-Gauss Kernel pro Pixel in horizontaler Richtung
2. Resultat aus 1. wird mit dem selben Kernel in vertikaler Richtung gefaltet
3. 1. und 2. werden mit verschiedenen Kernelgrößen so oft wiederholt, wie für die DoG-Pyramide nötig
4. Größe halbieren und zurück zu 1.

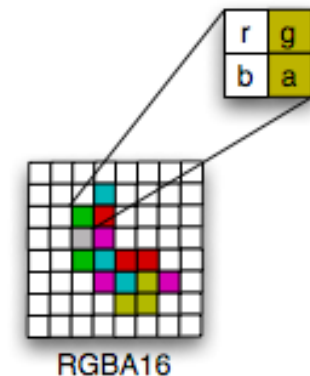
## Probleme/Chancen bei GPU Implementierung:

- Operationen mit Graubildern
- Verschenkte Rechenleistung durch skalare Operationen
- GPU rechnet auf Vektoren
- Ausnutzen von Vektorberechnungen für Gaußfaltungen notwendig

# DoG-Pyramide auf der GPU mit gepackten Daten



- Bild wird horizontal und vertikal "gepackt"
- pro: GPU kann besser genutzt werden
- contra: Anpassen von Algorithmen u.U. notwendig  
Zusätzlicher Rechenaufwand zum Packen
- Bildgröße, somit Ausführen des Fragment  
Programms nur 1/4

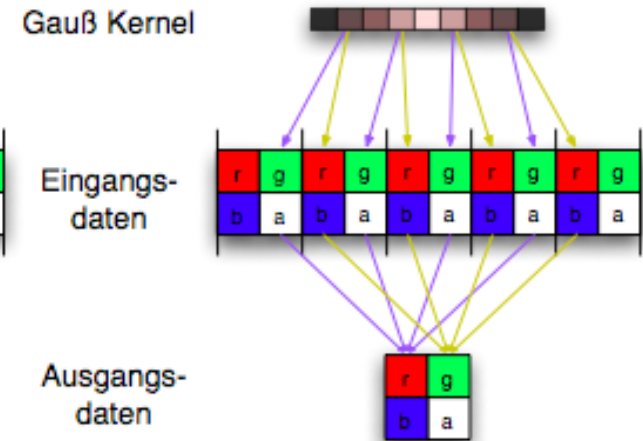
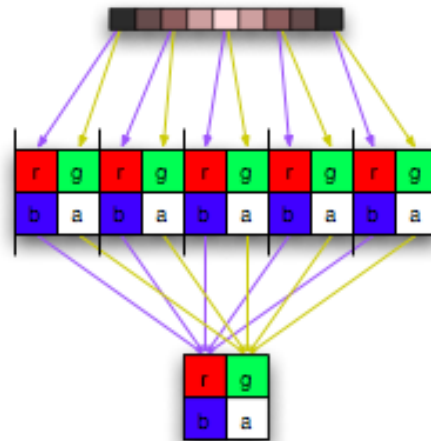
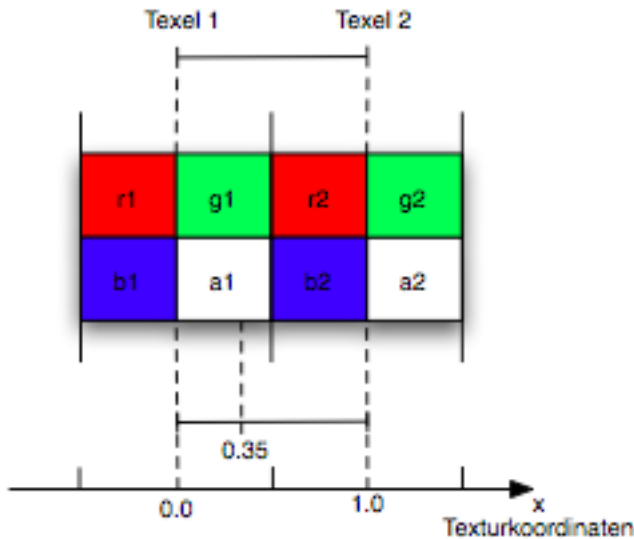


# Faltung gepackter Daten

- Gepackte Faltung in zwei Teilen
- Ausnutzung linearer Interpolation
- Pro:
  - Ideale Ausnutzung der Grafikkarte
- Contra:
  - Hoher Entwicklungsaufwand
  - wenig flexibel

Prozessortyp	Zyklen
CPU	26 ( 104 für vier Punkte)
GPU (ohne Packing/mit VLerp)	13 ( 52 für vier Punkte)
GPU (mit Packing/ohne VLerp)	46
GPU (mit Packing/VLerp/FLerp)	15

GPU Daten gemessen mit *nVidia's* cgc Compiler



# Erzeugen der *Difference of Gaussian* Pyramide

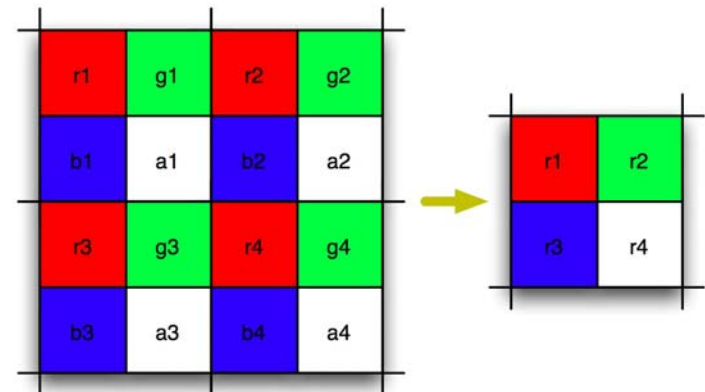
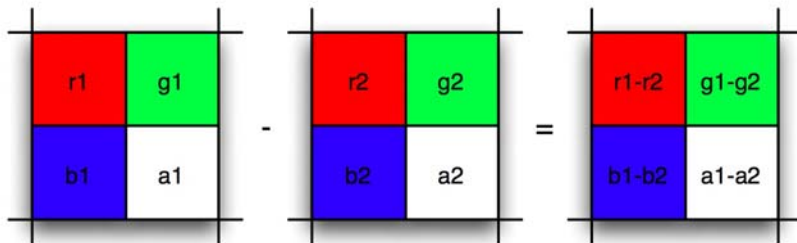
- Subtraktion der einzelnen

Faltungsstufen:

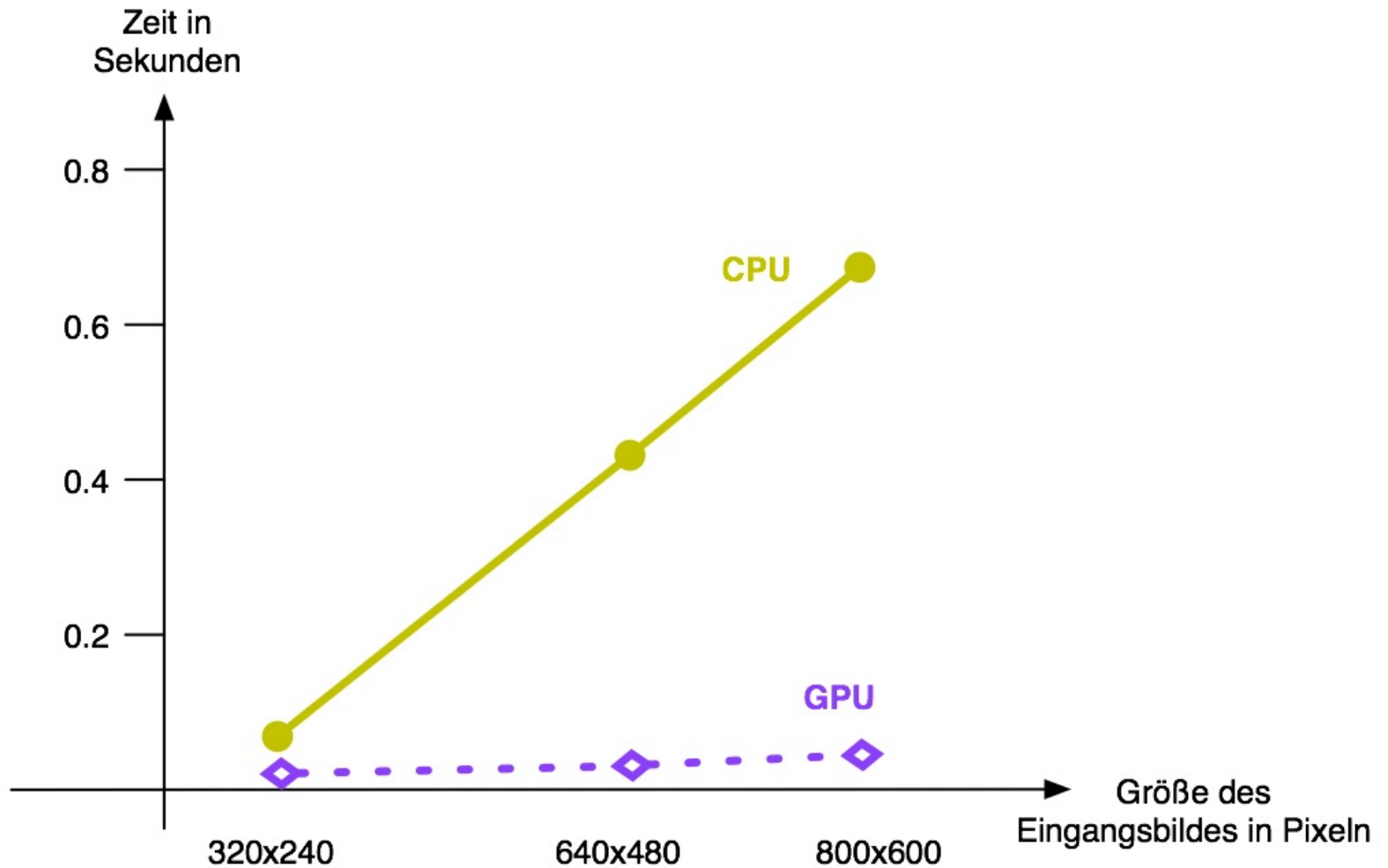
- Eins-zu-Eins Abbildung der Pixel zueinander
- gepackte Daten benötigen keine Extrabehandlung

- Reduktion der Bildgröße:

- Halbieren der Berechnungsgröße
- kein weiterer Informationsverlust aufgrund der wiederholten Tiefpassfilterung des Eingangsbildes

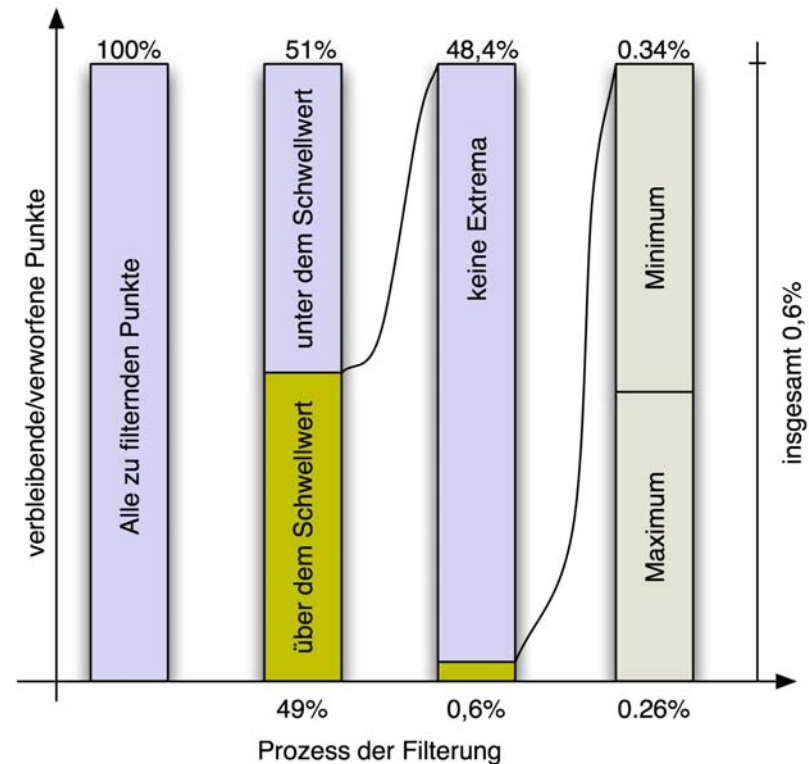
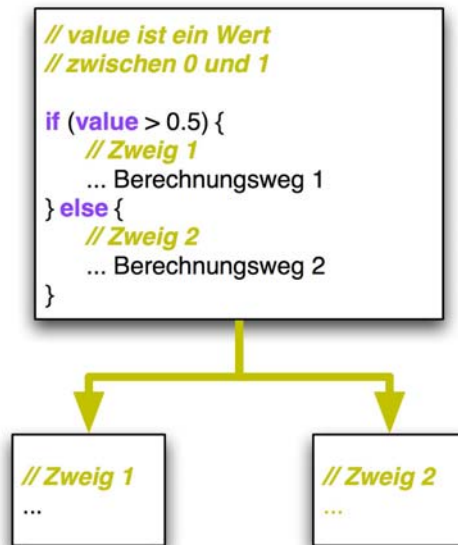


# Vergleiche



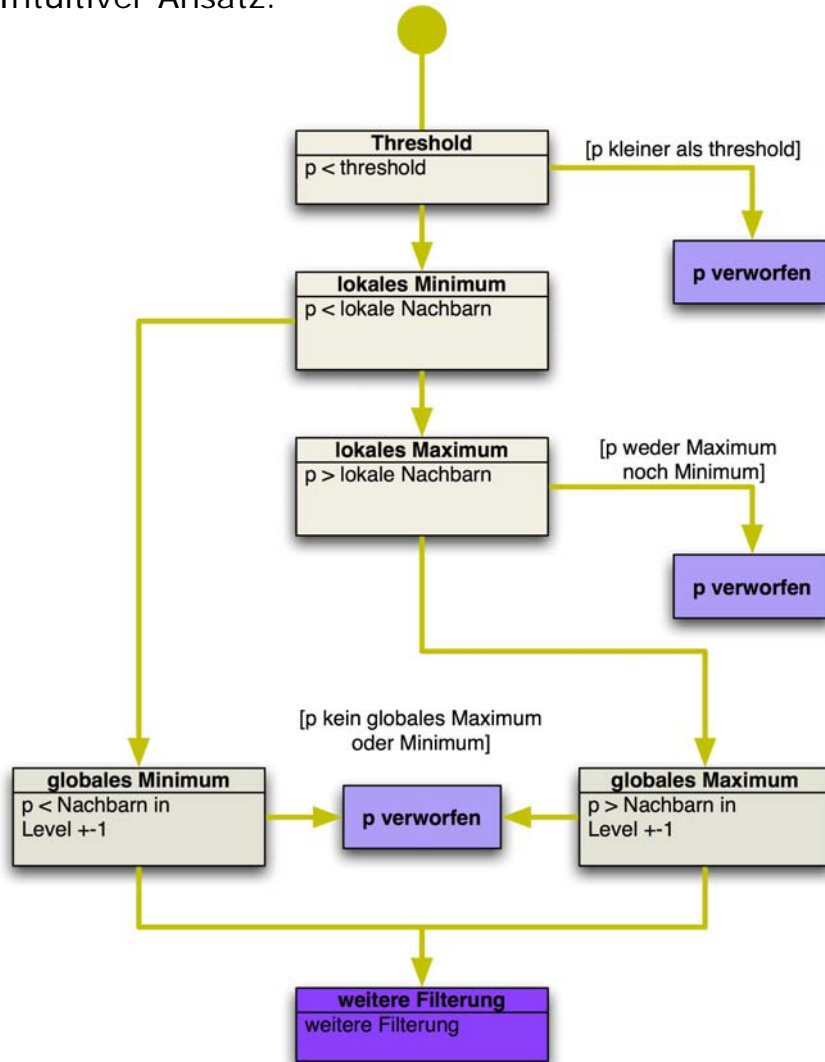
# Filterung der Punkte auf der GPU

- gestaffelte Filterung / verbleibende Punkte der vorangegangenen Filterschritte werden weiter berücksichtigt, andere verworfen
- Branch Prediction auf CPU macht Filterung effizient möglich
- gesamte Filterung in einem Rendering-Pass
- Analyse der einzelnen Filterstufen ermöglicht Verzweigungsoptimierung

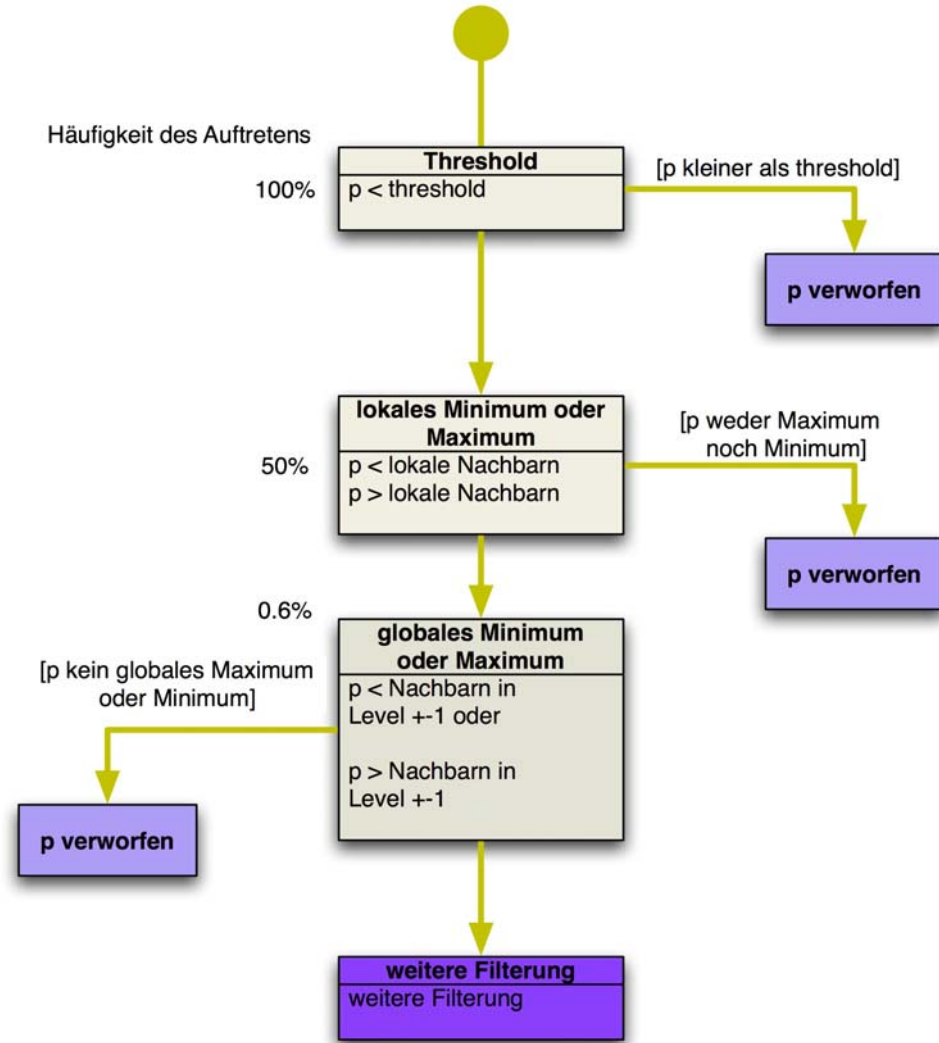


# Konditionalverzweigungen der Filterung

Intuitiver Ansatz:

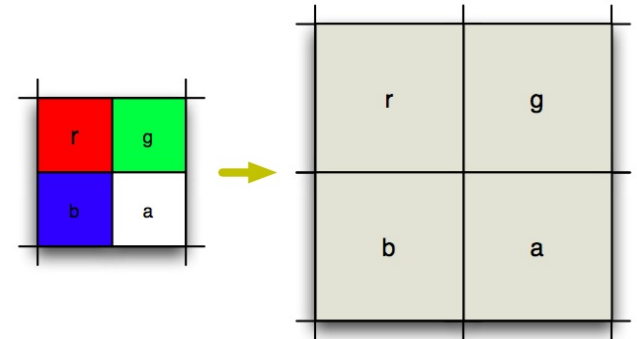
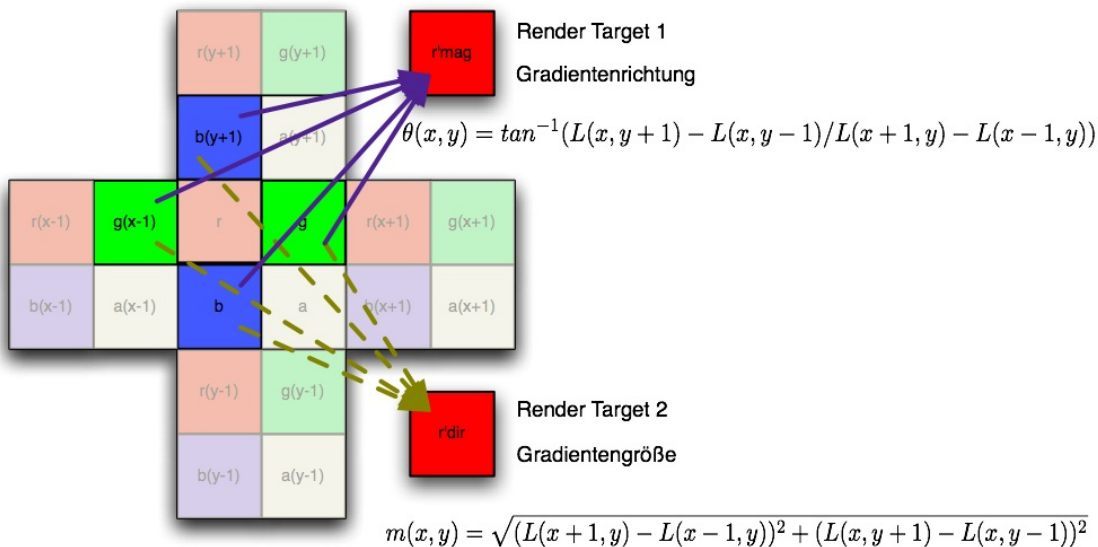


Branching optimierter Ansatz:



# Vorbereitung der Gradientengröße und -richtung

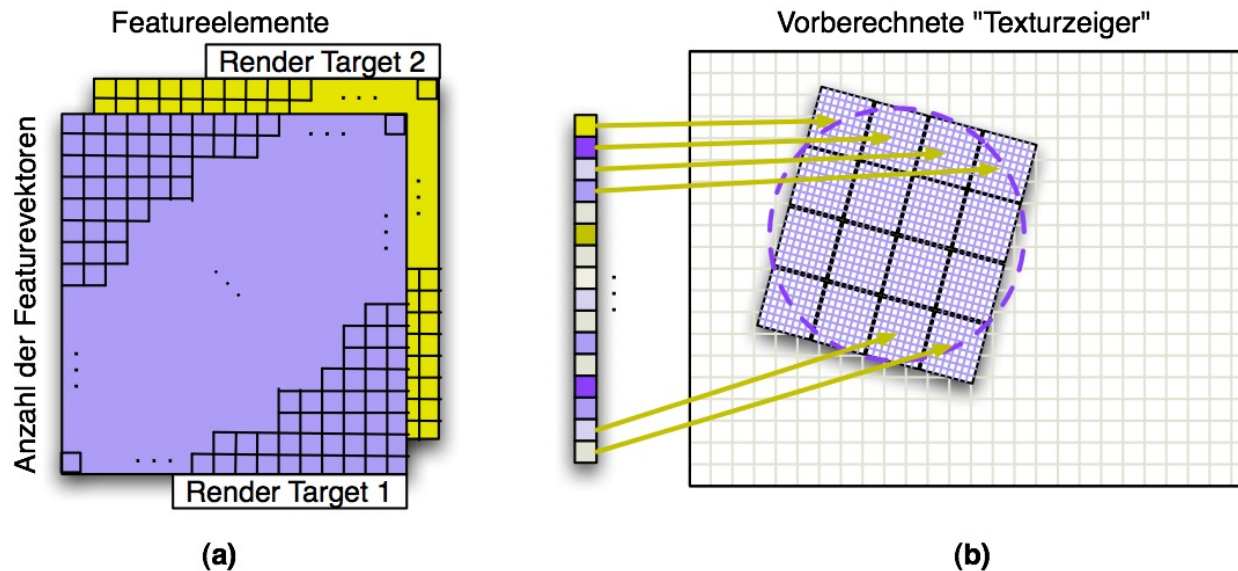
- Berechnungen auf gepackten Daten
- Eine Textur pro Skalenraumstufe
- Basis der Featurevektorberechnungen
- gepackte Daten ab jetzt ungünstig
- Entpacken der Daten
- Abbildung der RGBA Anteile in je einen Grauwert
- Später schnellerer Zugriff möglich



$$m(x, y) = \sqrt{(L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2}$$

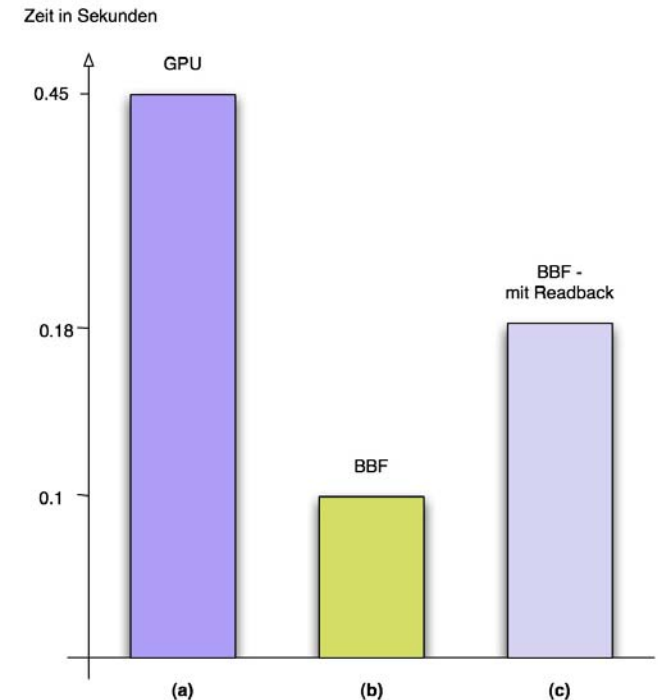
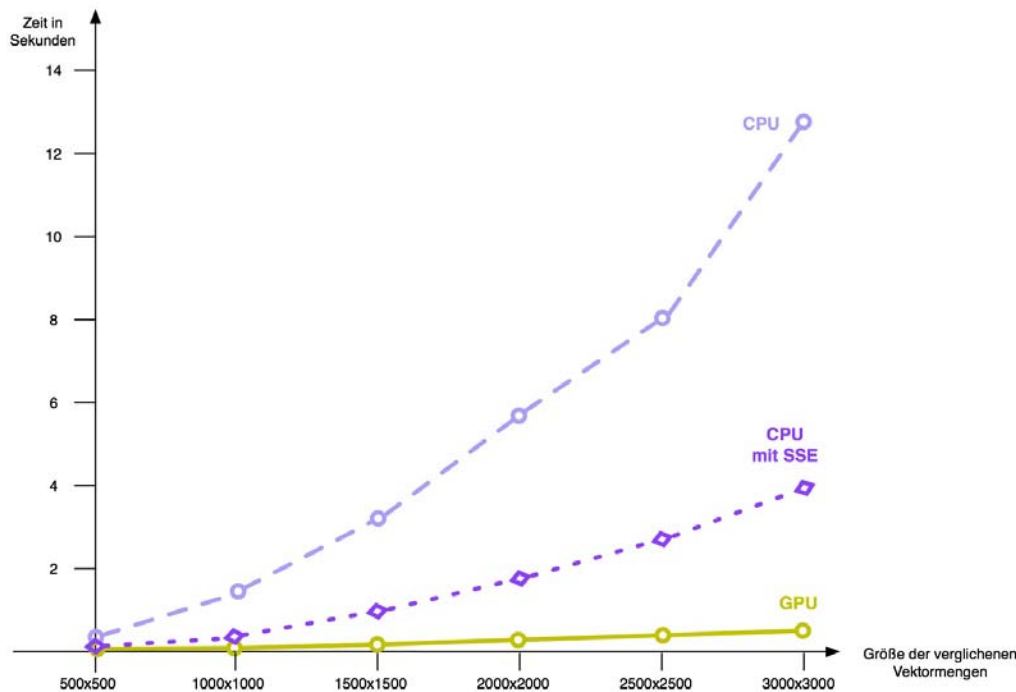
# Berechnung der Featurevektoren

- Gefilterte Extrempunkte liegen in einer Textur pro Skalenraumstufe vor
- Zurücklesen auf CPU notwendig
- Aufteilen auf 2 Rendering Targets
- Erzeugung von Geometriepunkten für Vektorelemente (1 Punkt je Fenster)
- Position, Rotation und Größe des Keypointfensters in Texturenkoordinaten



# Matching der Vektoren

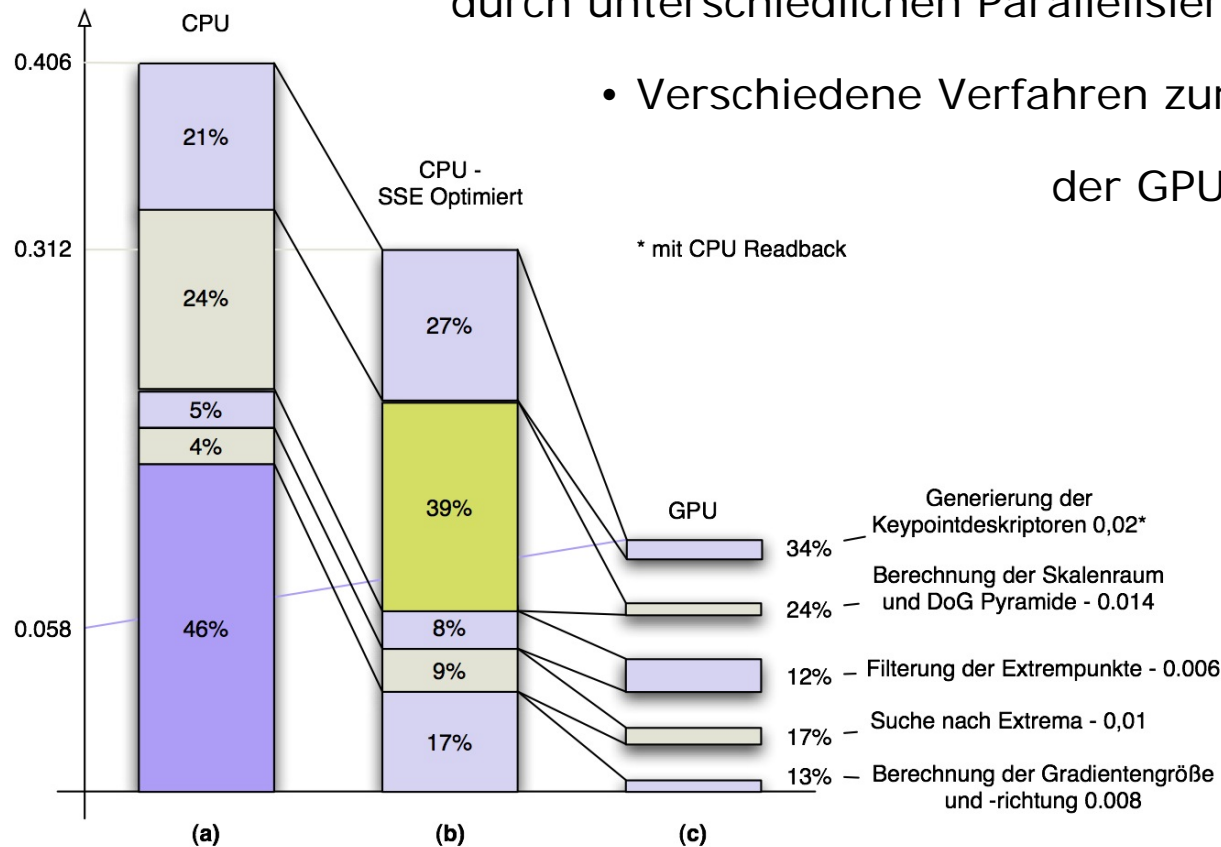
- Berechnung des euklidischen Abstandes
- Brute Force Implementierung liefert auf GPU gute Resultate, da sehr gut parallelisierbar
- Berechnungsfenster der Größe "*Referenzfeatures x Bildfeatures*"



# Resultate und Zusammenfassung

- Echtzeitbenutzung des *SIFT* Verfahrens auf GPUs möglich
- prozentuale Verschiebung der Berechnungsschwerpunkte durch unterschiedlichen Parallelisierungsgrad

Zeit in Sekunden



- Verschiedene Verfahren zur Ausnutzung der GPU-Fähigkeiten

# Ausblick

- Praktische Anwendung des Verfahrens
- durch GPU Nähe ist eine Anwendung in Augmented Reality denkbar
- Optimierung einzelner Schritte möglich
- Abwärtskompatibilität für *GeForce FX* Grafikkarten wäre denkbar
- Kameraeingabe für Live-Demos

# Danke