

„kinectboard“ — Objekt-Tracking mit CUDA und Kinect

Michael Stapelberg, Felix Bruckner, Pascal Krause

Fakultät für Informatik
Hochschule Mannheim

2012-07-19

Allgemeines

- Unser Projekt
- Vorgehen

Hardware

- Kinect
- Grafikkarte
- Handschuh

Algorithmen

- Median-Filter
- Kalibrierung
- Gloweffekt
- RGB-Bild maskieren
- Referenz-Farbe

Hardware-Rendering

Ausblick

Projektidee: Objekttracking

Jede einigermaßen ebene Fläche soll als Whiteboard dienen können.

- ▶ Schwierigkeiten
 - ▶ Pure Farberkennung zu viele false-positives
 - ▶ Qualität des Tiefenbildes zu schlecht
- ▶ Lösungsansatz
 - ▶ Aufbereitung der Daten durch Anwenden von Filtern

Vorgehen

- ▶ Suchen eines open-source SDKs
- ▶ Erforschen der Kinect
- ▶ Suchen nach Problemlösungen
- ▶ Validieren der Lösungen (CPU)
- ▶ Portieren der Lösungen auf GPU
 - ▶ Programmaufbau an CUDA anpassen
 - ▶ Algorithmen für CUDA optimieren

Kinect



- ▶ Sensoren
 - ▶ 640x480 30Hz Farbbild (RGB)
 - ▶ 640x480 30Hz Tiefenbild
- ▶ Versetzte Kameras
- ▶ Genauigkeit
 - ▶ Genauigkeit ab 50cm ca. 1,5mm
 - ▶ Genauigkeit ab 5m ca. 5cm

Grafikkarte



- ▶ Nvidia GeForce GTS 250
- ▶ 16 Multiprozessoren mit je 8 Cores
- ▶ 1024 MB Device-Memory

Handschuh



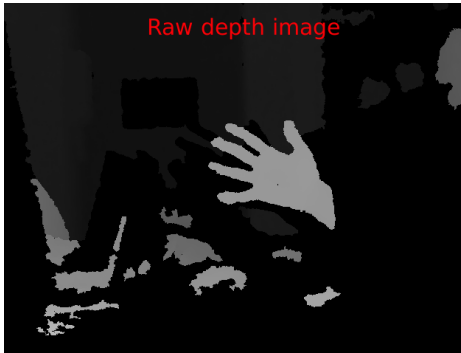
- ▶ 100% Baumwolle
- ▶ Hoher Tragekomfort
- ▶ Farbe: Orange

Median-Filter

- ▶ Arbeitet auf dem Tiefenbild
- ▶ Filtert das Rauschen aus dem Tiefenbild heraus

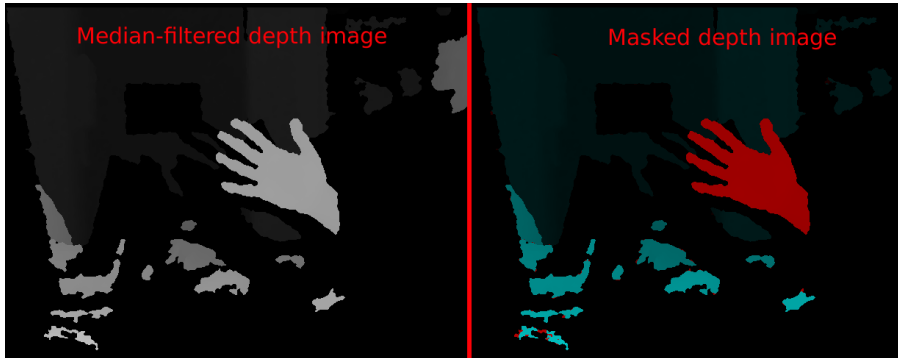
$-1, -1$	$0, -1$	$+1, -1$
$-1, 0$	i	$+1, 0$
$-1, +1$	$0, +1$	$+1, +1$

Median-Filter



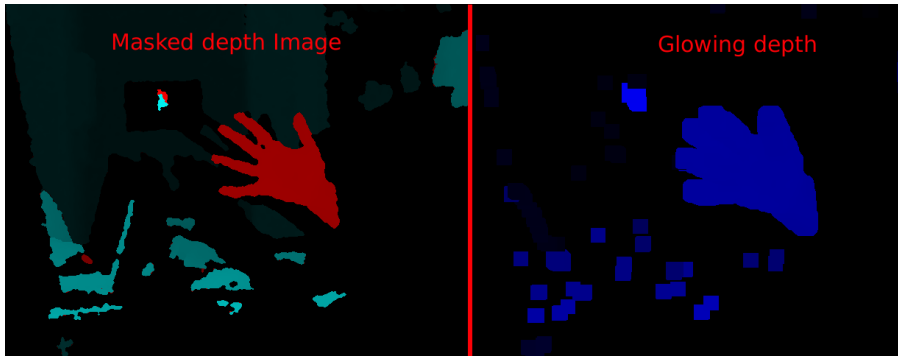
Kalibrierung

- ▶ Es kann auf neuen Hintergrund kalibriert werden
- ▶ Arbeitet auf dem Tiefenbild
- ▶ Filtert alle sich nicht bewegend Punkte aus dem Tiefenbild.



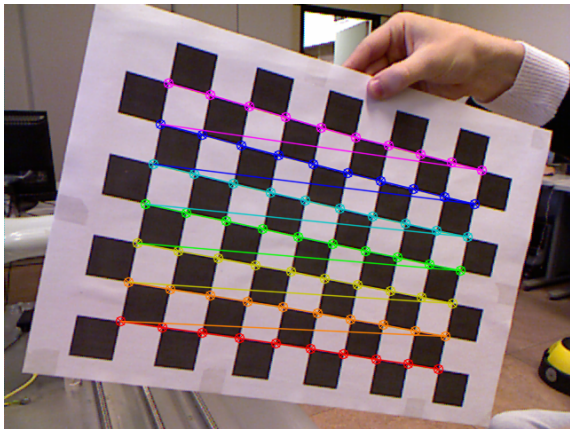
Glowlffekt

- ▶ Arbeitet auf dem Tiefenbild
- ▶ Expandiert alle anzuzeigenden Punkte im Tiefenbild, anhand eines einstellbaren Radius



RGB-Bild maskieren

- ▶ Arbeitet auf dem RGB-Bild
- ▶ Rechnet das Tiefenbild auf das RGB-Bild um, und maskiert alle relevanten Pixel



Referenz-Farbe

- ▶ Arbeitet auf dem RGB-Bild
- ▶ Zeigt nur noch die Pixel auf dem RGB-Bild an, welche farblich ähnlich genug zur Referenzfarbe sind



Median-Filter: Überblick

1. Host: Bilddaten auf die Grafikkarte kopieren, Kernel starten
2. Index berechnen, (umliegende) Pixel-Werte lesen
3. Median-Filter anwenden
4. Pixel-Wert umwandeln in RGB-Wert
5. Ausgabe-Pixel schreiben
6. Host: Bild anzeigen

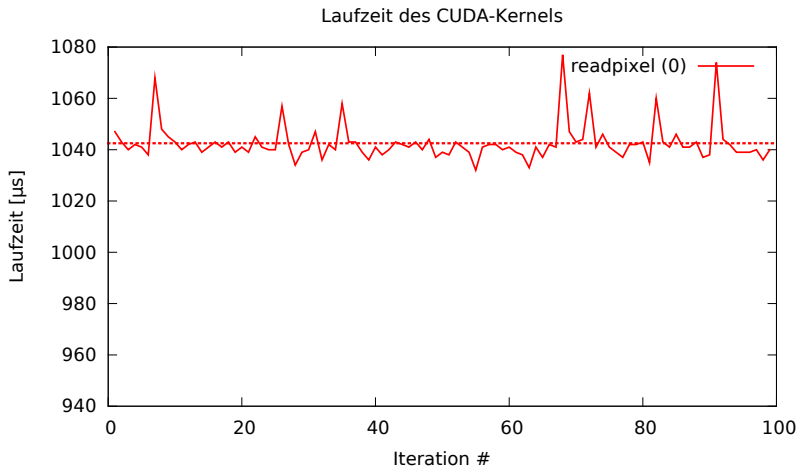
Wert lesen und umrechnen (readpixel0)

```
__global__ void median(uint16_t *depth,
    uint8_t *table, uint8_t *output)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;
    int i = (y * 640) + x;

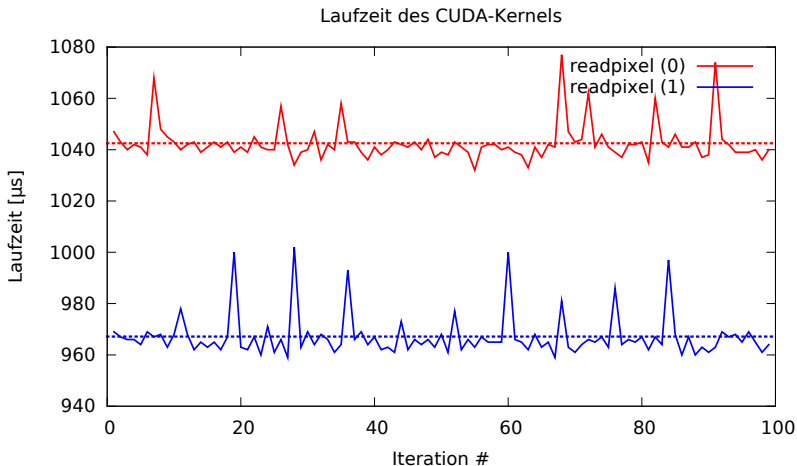
    // Pixel in RGB wandeln
    int conv = table[depth[i]];

    // Makro, welches output[i], output[i+1] und
    output[i+2] setzt.
    pushrgb(output, i, conv);
}
```

Wert lesen und umrechnen (readpixel0)



Wert lesen und umrechnen (readpixel0/1)



Wert lesen und umrechnen (readpixel1)

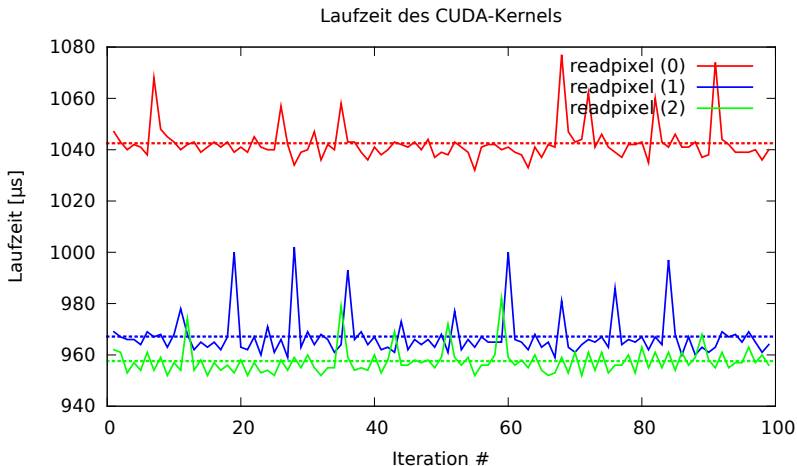
```
texture<uint16_t, 2> depthT;  
texture<uint8_t, 2> tableT;
```

```
__global__ void median(uint8_t *output) {  
    // x, y, i wie zuvor  
    int c = tex2D(tableT, tex2D(depthT, x, y), 1);  
    pushrgb(output, i, c);  
}
```

```
// memcpy table und depth
```

```
cudaChannelFormatDesc desc =  
    cudaCreateChannelDesc<uint16_t>();  
cudaBindTexture2D(NULL, &depthT, depth, &desc,  
    640, 480, 640 * sizeof(uint16_t));  
desc = cudaCreateChannelDesc<uint8_t>();  
cudaBindTexture2D(NULL, &tableT, table, &desc,  
    2048, 1, 2048 * sizeof(uint8_t));
```

Wert lesen und umrechnen (readpixel0/1/2)



Wert lesen und umrechnen (readpixel2)

```
texture<uint16_t, 2> depthT;  
texture<uint8_t, 2> tableT;
```

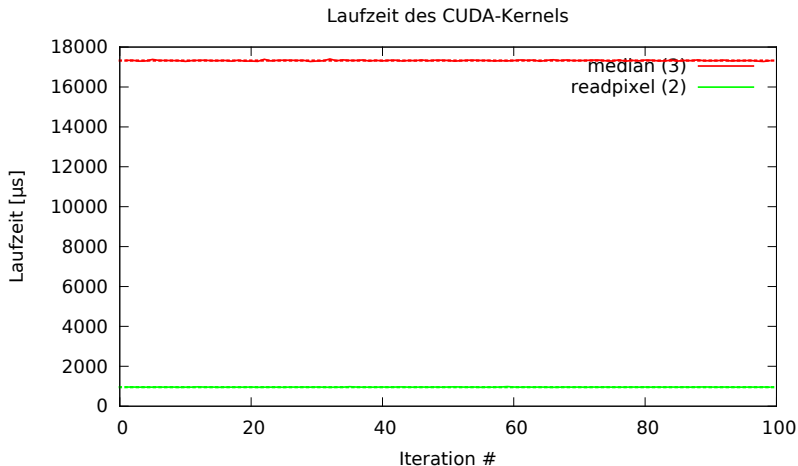
```
__global__ void median(uint8_t *output) {  
    // x, y, i wie zuvor  
    int d = tex2D(depthT, x, y);  
    int c = (float)(2048 * 256) / (d - 2048);  
  
    pushrgb(output, i, c);  
}
```

```
// memcpy table und depth  
// cudaBindTexture2D wie zuvor
```

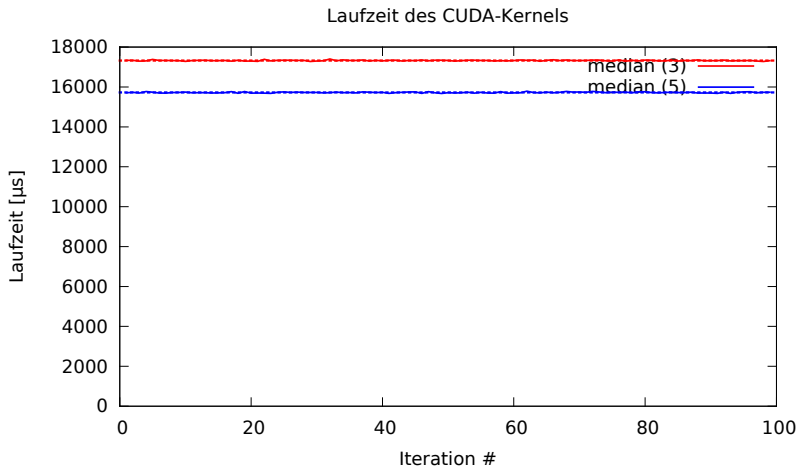
Median-Filter (median3)

```
__global__ void median(uint8_t *output) {  
    // x, y, i wie zuvor  
    int neigh[25], ic, ir, ni = 0;  
    for (ic = (x - (5 / 2));  
         ic <= (x + (5 / 2));  
         ic++) {  
        for (ir = (y - (5 / 2));  
             ir <= (y + (5 / 2));  
             ir++) {  
            neigh[ni++] = tex2D(depthT, ic, ir);  
        }  
    }  
  
    // quick_select aus "Numerical recipes in C"  
    int c = (float)(2048 * 256) / (quick_select(  
        neigh, 25) - 2048);  
    pushrgb(output, i, c);  
}
```

Median-Filter (median3)



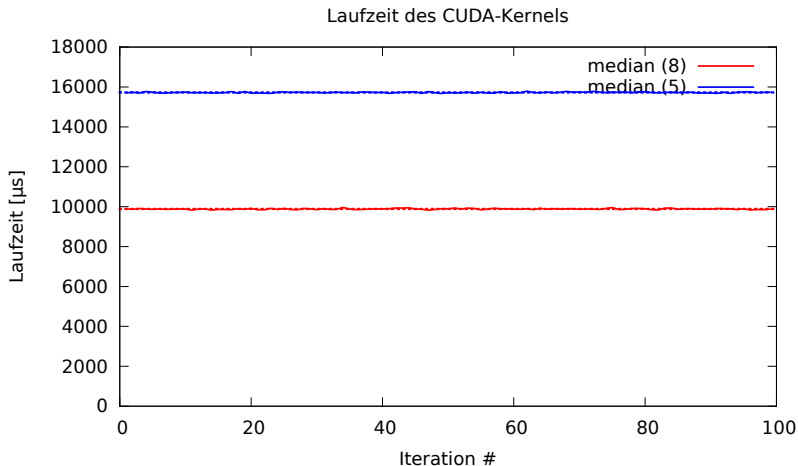
Median-Filter (median3/5)



Median-Filter (median5)

```
__global__ void median(uint8_t *output) {  
    __shared__ float smem[BLOCK_X][BLOCK_Y];  
    smem[threadIdx.x][threadIdx.y] = tex2D(depthT,  
        x, y);  
    __syncthreads();  
  
    int neigh[25], ic, ir, ni = 0;  
    for (ic = (threadIdx.x - (5 / 2));  
        ic <= (threadIdx.x + (5 / 2));  
        ic++) {  
        for (ir = (threadIdx.y - (5 / 2));  
            ir <= (threadIdx.y + (5 / 2));  
            ir++) {  
            neigh[ni++] = smem[ic][ir];  
        }  
    }  
  
    // quick_select wie zuvor  
}
```

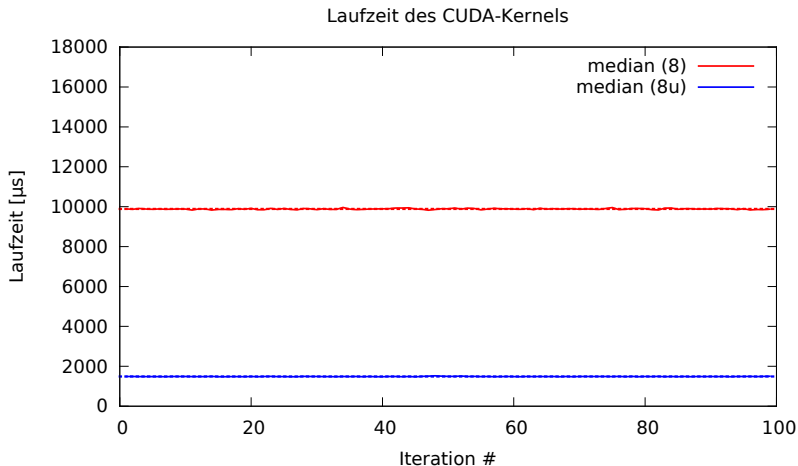

Median-Filter (median5/8)



Median-Filter (median8)

```
__global__ void median(uint8_t *output) {  
    int ic, ir, ni = 0;  
    float neigh[9], ma, mi;  
    for (ic = (x - 1); ic <= (x + 1); ic++) {  
        for (ir = (y - 1); ir <= (y + 1); ir++) {  
            neigh[ni++] = tex2D(depthT, ic, ir);  
        }  
    }  
    for (int i = 0; i < 8; i++) {  
        for (int j = 0; j < (8-i); j++) {  
            ma = fmaxf(neigh[j], neigh[j+1]);  
            mi = fminf(neigh[j], neigh[j+1]);  
            neigh[j] = mi;  
            neigh[j+1] = ma;  
        }  
    }  
    int c = (float)(2048 * 256) / (neigh[9/2] -  
        2048.0);  
}
```

Median-Filter (median8/8u)



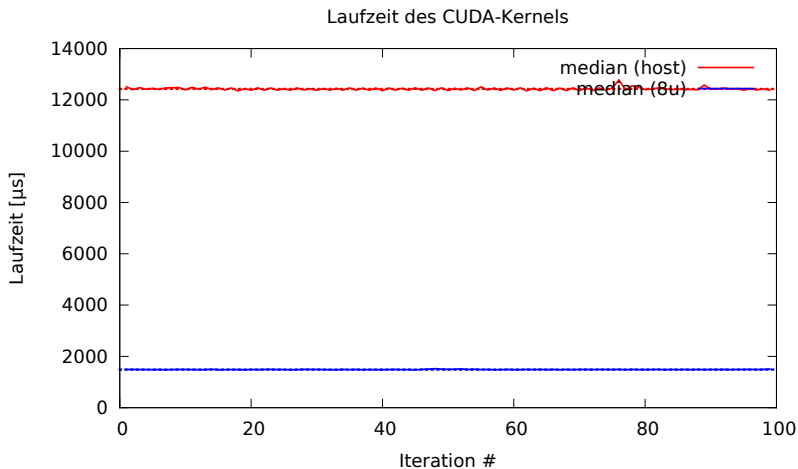
Median-Filter, PTX (median8)

```
$Lt_0_5634:
    <i>
```

Median-Filter (median8)

```
__global__ void median(uint8_t *output) {  
    int ic, ir, ni = 0;  
    float neigh[9], ma, mi;  
    for (ic = (x - 1); ic <= (x + 1); ic++) {  
        for (ir = (y - 1); ir <= (y + 1); ir++) {  
            neigh[ni++] = tex2D(depthT, ic, ir);  
        }  
    }  
    for (int i = 0; i < 8; i++) {  
        for (int j = 0; j < 8; j++) {  
            ma = fmaxf(neigh[j], neigh[j+1]);  
            mi = fminf(neigh[j], neigh[j+1]);  
            neigh[j] = mi;  
            neigh[j+1] = ma;  
        }  
    }  
    int c = (float)(2048 * 256) / (neigh[9/2] -  
        2048.0);  
}
```

Fazit



Speedup durch CUDA: $\approx 12\times$

Portierung auf GPU

- ▶ Ausgangssituation (CPU)
 - ▶ SDL Surface
 - ▶ Softwarerendering
 - ▶ 3 FPS bei einschalten der Filter
- ▶ Ziel (GPU)
 - ▶ Bilder von Host auf Grafikkarte kopieren
 - ▶ Bilder auf Grafikkarte berechnen
 - ▶ Bilder von Grafikkarte auf Host kopieren
 - ▶ extrem langsam

Optimierung 1

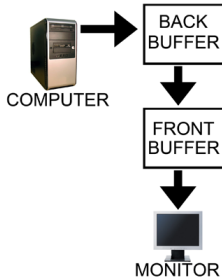
- ▶ Alle Bilder in einen Block
- ▶ Weniger Memcopys
- ▶ Immer noch zu langsam (5fps)
- ▶ Kosten für ein Memcopy 7ms
- ▶ Maximale Durchlaufdauer 30ms

Optimierung 2

- ▶ Hardwarerendering mit OpenGL
- ▶ Spezielle OpenGL-Bufferobjekte
- ▶ Nur noch kopieren der Input-Bilder erforderlich

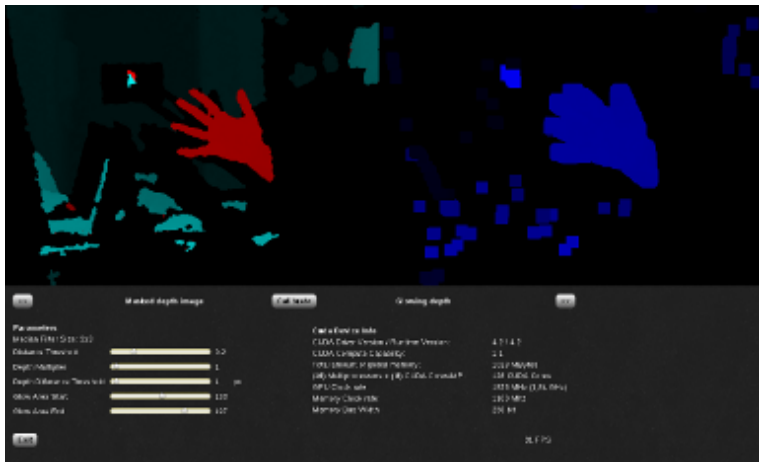
Optimierung 3

- ▶ Doublebuffering verhindert flackern
- ▶ BufferObjekt besteht aus front & back
- ▶ Bufferwechsel bei Pixelveränderung
- ▶ VSync



GPU GUI

- ▶ Alte SDL-GUI keine OpenGL unterstützung
- ▶ Awesomium



Live Demo

Live Demo

Ausblick

- ▶ Bewegung interpolieren
- ▶ Extremitäten statt Pixel
- ▶ Visuell bedienbare GUI
 - ▶ Buttons
 - ▶ Gesten
- ▶ Ausführlichere Dokumentation
- ▶ Plattformabhängigkeit minimieren

Vielen Dank
für Ihre Aufmerksamkeit