

geofront

VISUAL COMPUTING

geofront is an Austrian R&D company for computer vision and general purpose computation on graphics hardware (aka GPUs).

Line Segment Maps: Fast and Deterministic Line Detection on the GPU (Dr-Ing. Gernot Ziegler)

We develop novel cutting-edge algorithms, applying real-time graphics concepts to computer vision tasks and other spatial signal processing, such as light wavefront simulation, video compression and computer tomography.

We supply contracted research and software development, leading engineers to implemented solutions. Services include task assessment and white board advice on solution approaches in workshops. These can also be combined with training for GPU development.

Finally, our on-demand advice supports your engineering staff during their graphics

Business Background

- geofront e.U., privately owned, was founded in January 2016.
- Algorithmic consulting and training in general purpose GPU computing.
- Main focus on **computer vision** and visual computing, e.g. GPU-based camera calibration or lightfield acquisition/rendering.
- Spatial computing in HPC as well, e.g. volume compression or tomography reconstruction.

geofront
VISUAL COMPUTING

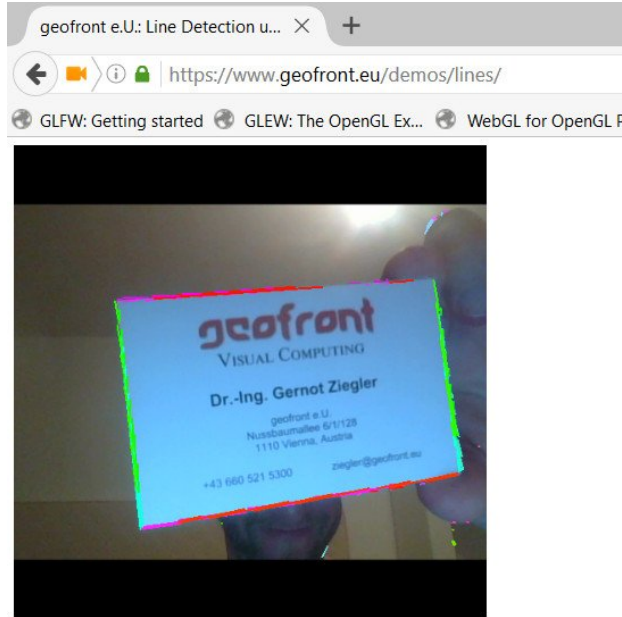


Open Investigations: Purpose

- Create new algorithms and technologies as part of geofront's algorithmic portfolio.
- Real-time game graphics techniques (on GPU, e.g. shadow mapping, Z buffer, data parallelism)
- Computer vision knowledge on multi-view acq. vision (Image Based Reconstruction and Rendering)
- Client-Server and Mobile Rendering / Vision (WebGL, Android, but also Tegra X1)

Line Detection in WebGL

<https://www.geofront.eu/demos/lines>



Line segments derived from
local edge filter results (line angle bins)

Data-parallel Segmented Scan
„connects“ local edges

2D line segment maps, one per line angle bin

Interactive run times on mobile GPUs

Novel for WebGL, through

Data Compaction algorithm which produces lists
of line segments above certain length

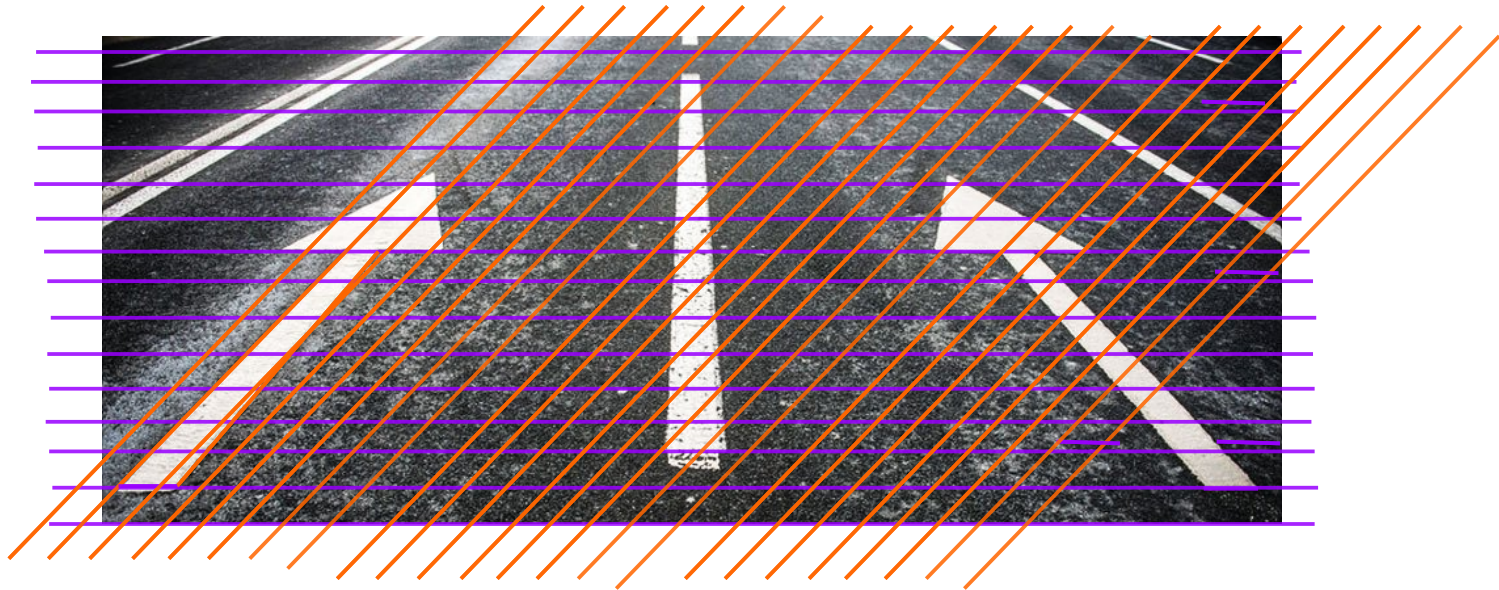
Task Setting

- Reliable Real-time Line Detection in images and video (a basic component for medium-level vision, e.g. in cars)
- Lines can be at *any angle*. Line start/end is important, too.
- Example input in automotive computer vision:



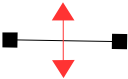


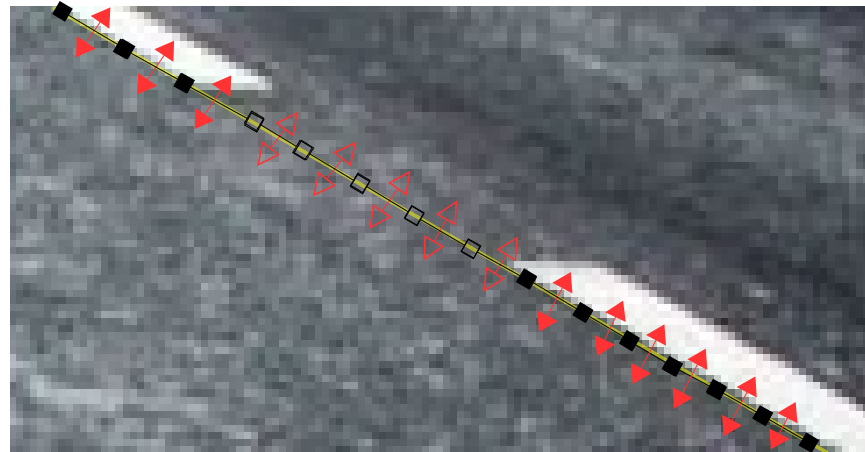
Line Hypotheses

- Line hypotheses are „laid“ at all aftersought angles
- Best to maintain input image resolution (avoid undersampling)
- Also parts of the hypothesis can be true



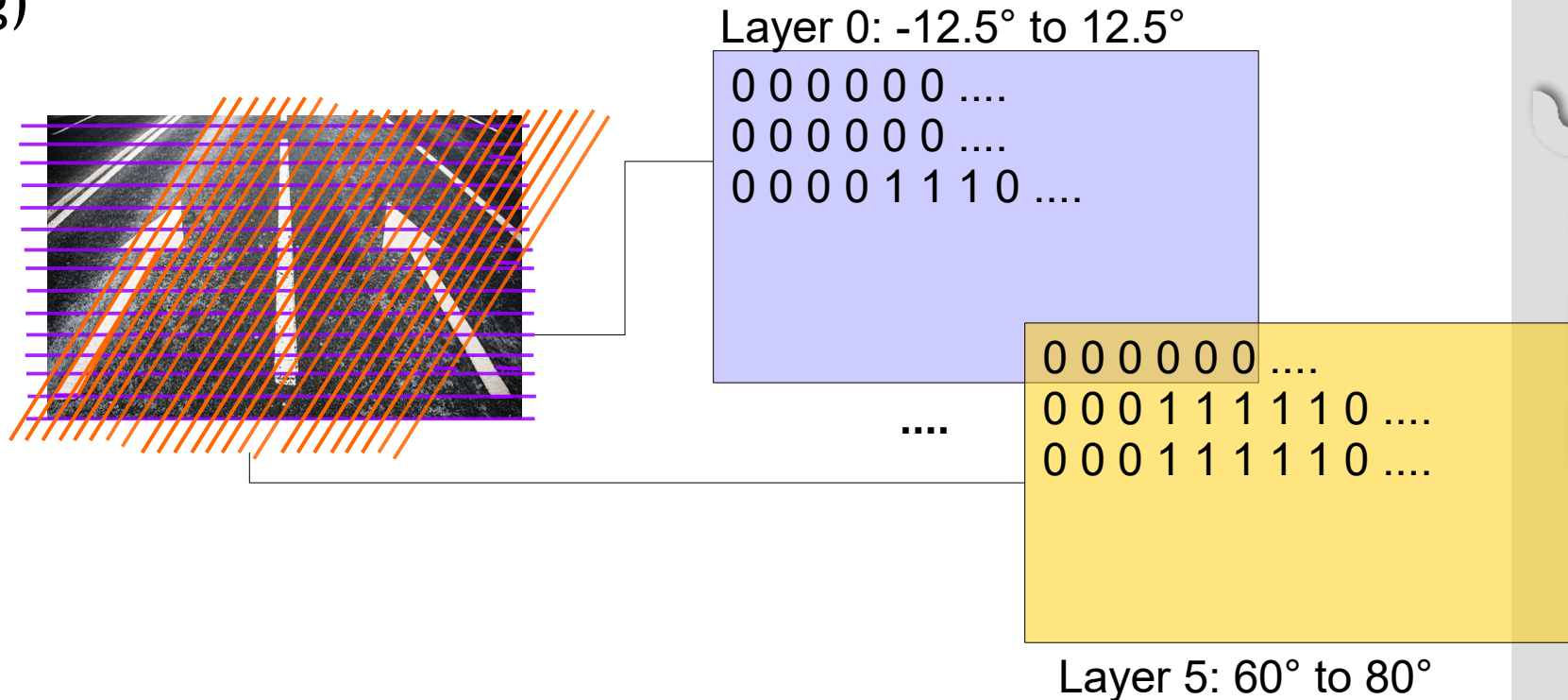
Line Atoms, or: What is a „line“?

- „Line segment hypotheses“ verified by **edge detection** (**central difference thresholding**)
- Condition 1: No edge along line hypothesis (yellow) 
- Condition 2: Edge present orthogonal to line hypothesis 
- **Line atom:** 
One local verification,
both conditions met.



Line Segment Map

- We create a 2D map over all line atom tests (0:s and 1:s)
- Layers represent the angle bins (e.g. Layer 0: Angle 0-22.5 deg)



Challenges

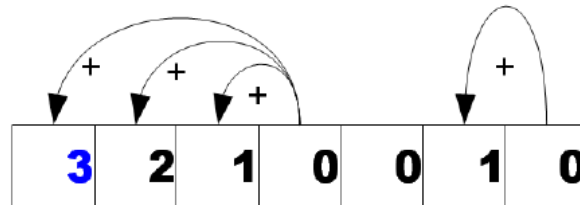
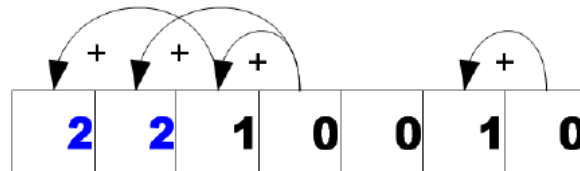
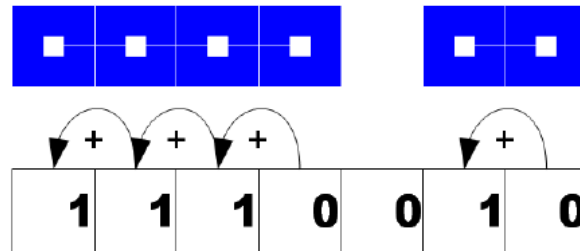
- Massive memory bandwidth needed:
Edge detection has to read pixel values repeatedly to evaluate line hypotheses at all angles.
- Edge detection at non-trivial angles
requires bilinear interpolation of input pixels.
- **On GPU: Texture cache and shared memory alleviates this!**
- (On CPU: Must often resort to stochastic evaluation - but makes line detection is *non-deterministic*)

Inference of Line Segments

- Now that we have found line support through edge detection, how to infer line segments? CPU approaches use Edge Tracking, but that is serial.

- We use segmented scan:

From 2 and 2 connected elements, we can infer that all 3 are connected, etc.

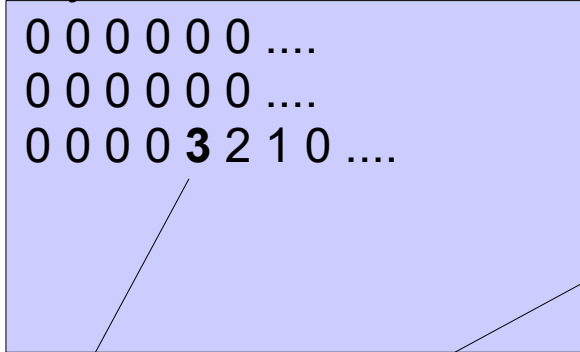


- CUDA, OpenGL Compute: Shared memory
- OpenGL ES 2.0, WebGL: Repeated shader calls

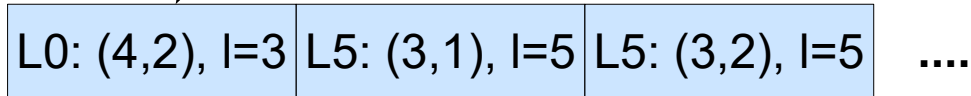
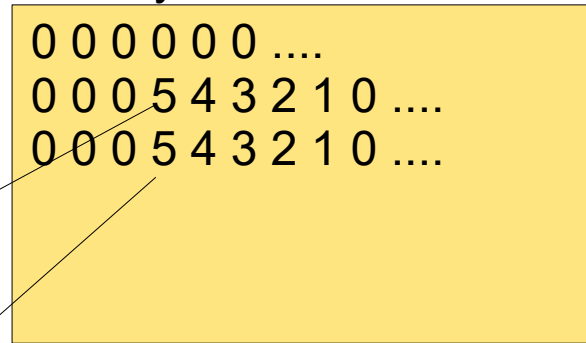
Line Segment Map to Line List

- Inference stores lengths in Line Segment Map
- Create Line list (*) via „Values above Threshold T, preceded by a Zero“

Layer 0: -12.5° to 12.5°



Layer 5: 60° to 80°



- (*) OpenGL or CUDA : Atomics (OpenGL ES, WebGL: Data Compaction, postponed)

Result

- Line List can now be rendered or used in further processing

L0: (4,2), l=3 | L5: (3,1), l=5 | L5: (3,2), l=5

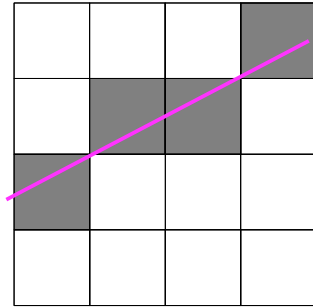


- (Line Segment Map is still useful, showing later)

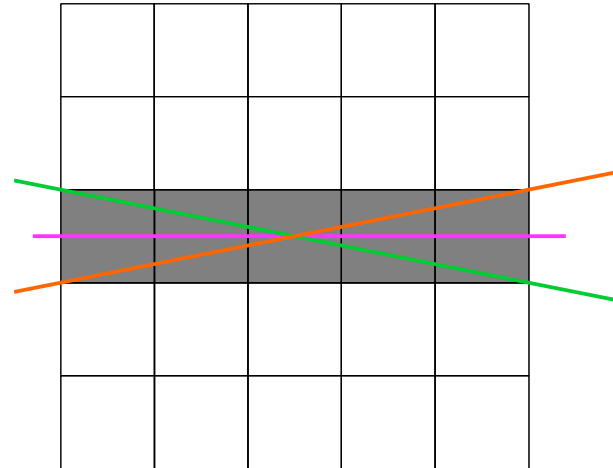
Musings on detection guarantees

- How can we make sure that every line direction angle is detected? (And only falls into one angle bin?)

- Look at Bresenham algo:
(rasterizes lines based
on their slope angle):

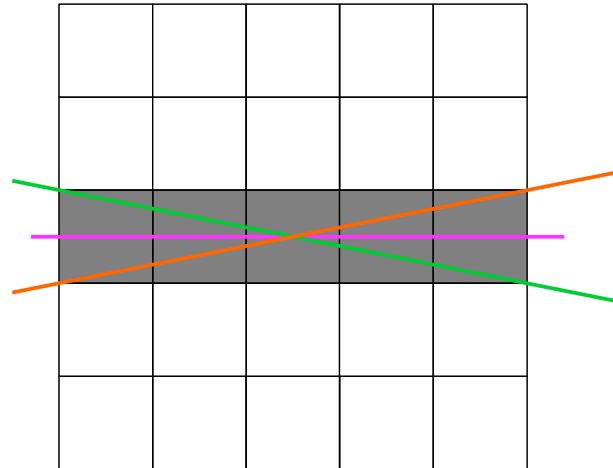


- Insight:
For short line segments,
close line angles cannot be told apart
- (they have identical Bresenham images).**



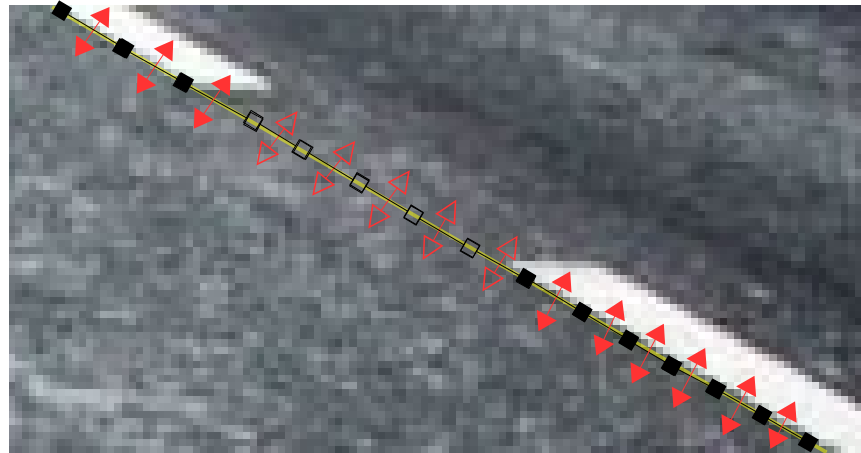
Musings on detection guarantees

- How many angle bins needed to catch all angle?
(Without proof: Number of required angle bins is outer rim cells of Bresenham window: here, 16 angle bins (easy to count), or 8 if 180 degrees symmetric.)
- Larger Bresenham window : Narrower angle range
- Gerneal:
The longer found
line segments are,
the more we can tell about their angle!
- Optimization : Use multi-stage to catch all angles;
Then you can subdivide angle bins recursively to narrow down.

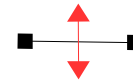


Angle identification in real images

- Reality has smaller image gradients than those from black-on-white Bresenham lines:



- But: Central difference cross for any pixel pattern responds highest on „its“ matching angle bin -> Proper detection threshold can be determined (unless crosstalking is desired for ambigious cases).

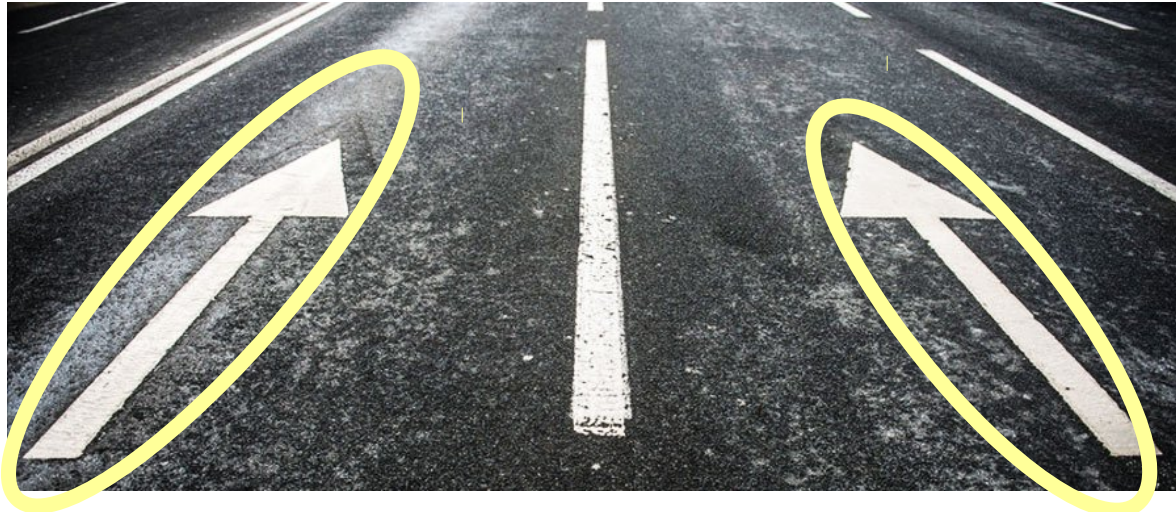


Imperfection tolerance

- Segmented scan expects line atoms at every sample position of line hypothesis.
- But lines in input image might be imperfect (e.g. withered road marking).
- **Fill-in** step before seg-scan starts.
- E.g. allow for one missing line atoms: **insert 1 iff 1 before and after.**
0 0 1 1 1 0 1 1 0 0 -> 0 0 1 1 1 1 1 1 0 0
- **BUT:** How long defects do we want to tolerate?
-> We might connect line segments that should not. Use dep. on application.
- (Note: stochastic approach might ignore imperfection -
but that is not deterministic!)

Primitive Detection

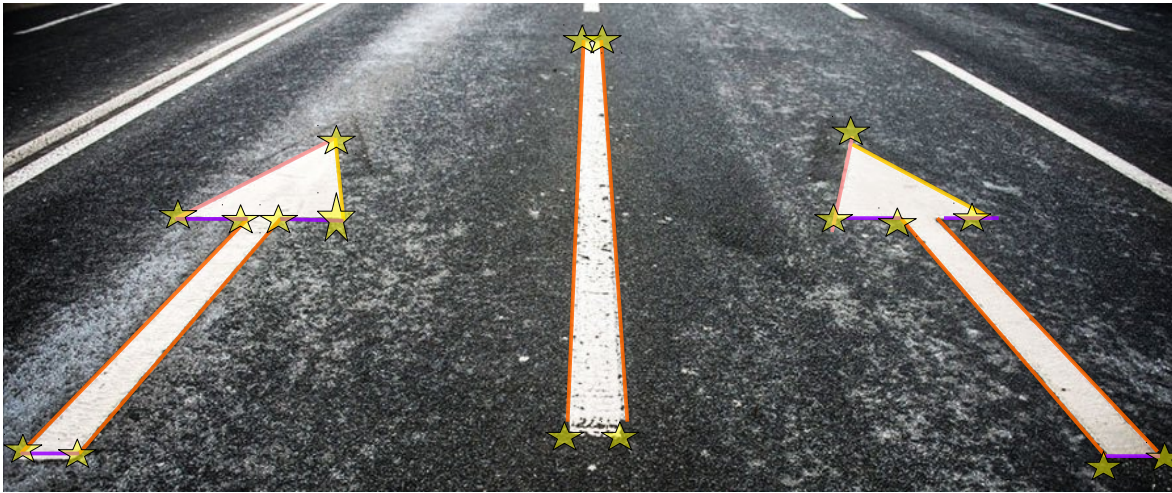
- Complex symbols are needed by high level AI (e.g. traffic signs or road markings)
- Searching through whole image is *prohibitively expensive* (esp. considering symbol's possible rotation angles and scales)



- How can we reduce bandwidth and computation requirements?

Primitive Detection

- Look first for components of the higher level primitives: Lines!
- Only „deeper“ investigation where lines have been found: ★



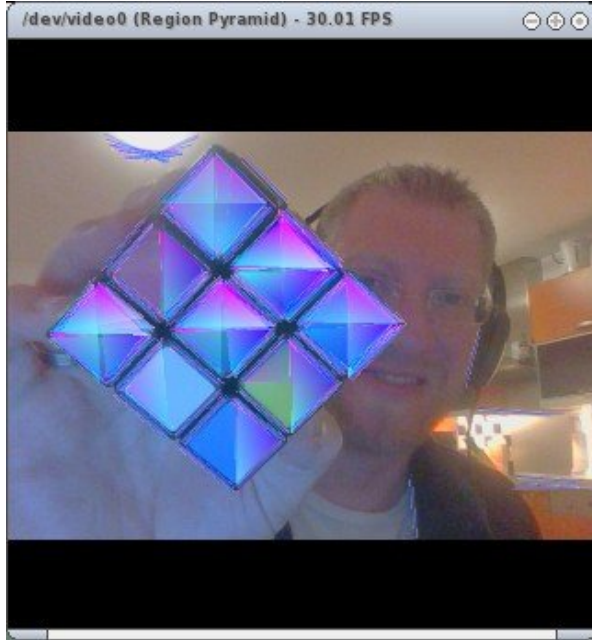
- Line list -> Massively reduced computation & bandwidth

Example: Quadrangle Detection

- We want to look for quadrangles of a side length min. 50 pixels, made up of (approx.) orthogonal lines (angle 70-110 degrees)
- First, we create a list for all lines at 50 pixel min length in the image (side result: Line segment map)
- Start a thread for every line
- For every thread: use line origin coordinate to look up in **line segment map** if there are other lines originating at same position (tolerance window, e.g. +/- 3 pixels in x and y).
- Effect: Maintain spatial correlation of lines via line segment map, but benefit from reduced parallelism of line list to detect quads.

Primitives from Line Detection (OpenGL, coming in WebGL)

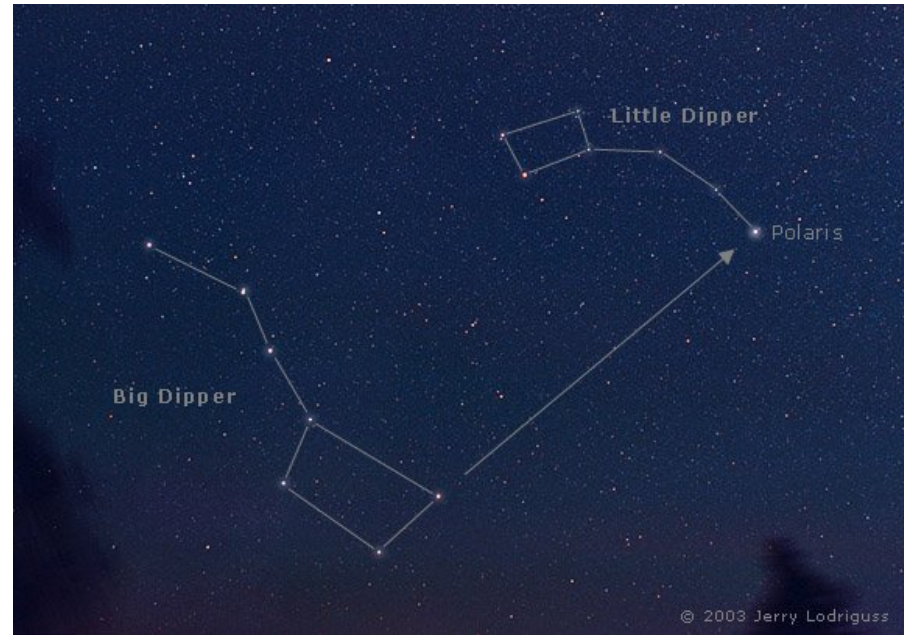
May 2016



- Higher level primitives (e.g. parallel lines, quadrilaterals, vector symbols, ...) usually too complex to search for in whole image (scale, rotation, ...)
- Data compaction reduces candidates to line segments of required length (as parts of above primitives)
- Side results of 2D line segment maps (one per line angle bin) still provides spatial neighbourhood info
- Correlation only done in relevant spots (with tolerances)
- **Augmented Reality Marker Tracking in WebGL browser, mobile and desktop, without .exe**
- **Computer Vision tasks on limited OpenGL ES 2.0 hardware (e.g. automotive, robotics, ...)**

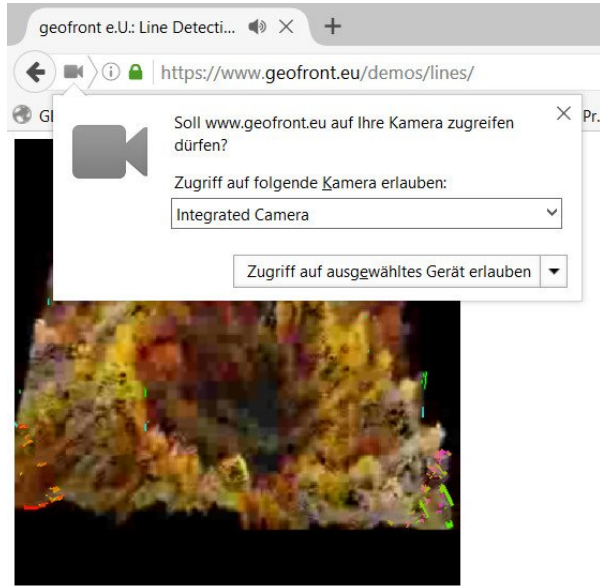
Primitive detection in HPC signal processing

- This can be generalized, detecting complex shapes that are comprised of subshapes of similar „atoms“. Atom and sub-shape identification limits complexity of shape search to promising locations. Shapes can be built in multiple stages that limit complexity every stage.
- Example from astronomy:
Stars as atoms
Constellations as Symbols.
- Multi-stage
Detect atoms (stars),
then two subshapes of
“The Great Bear“, then see if
the candidates comprise final.
- N-dimension as well!



Other Projects

Projects „WebGL Computer Vision,,



- HTML5 enables real-time video input
- WebGL enables GPU access from Javascript
- Limited (only OpenGL ES level), but 2008 algos for data compaction apply!
-> can *extract sparse feature lists*, build quadtrees, octrees, **geometry shader**, etc.
- Line detection, Object tracking, ...
- Contact me if you are curious and I give you a short intro!

Bounding Box Tracking in WebGL

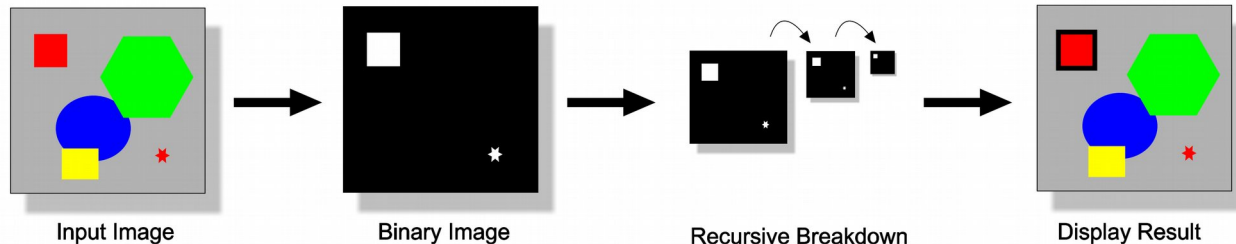
<https://www.geofront.eu/webgl/bboxpyramid>



- Quick tracking of largest pixel group of certain color
- Threshold pixels, assume small bounding boxes
- Data Parallel Reduction:
Merge bounding boxes (if adjacent) OR
Choose largest one (if competing)

Interactive run times on mobile GPUs

Doesn't have to be color (e.g. group local motion vectors)



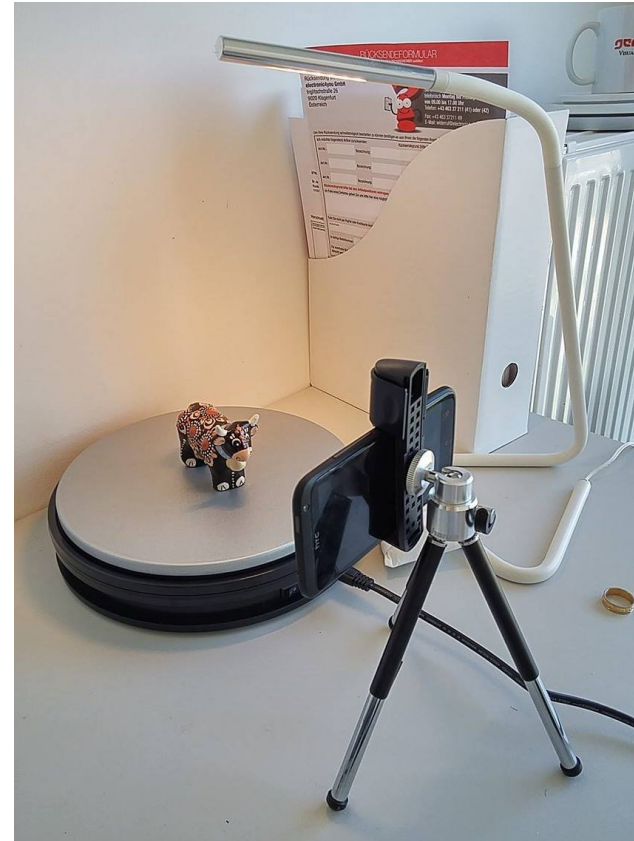
Photogrammetry

- Reflective materials and their lighting are hard to reproduce in 3D rendering
- 3D Reproduction with light fields, i.e. dense view acquisition.

(2D Video Compression for elimination of redundancy - later depth maps with proj. texture mapping for compression and view angle interpolation)



<http://www.geofront.eu/demos/360rotate>



Other data-parallel research

- HistoPyramids have been extended to create quadtrees and octrees through bottom-up analysis:

www.geofront.eu/thesis.pdf

- Summed Area Ripmaps fix precision issues of Summed Area Tables / Integral Images:

<http://on-demand.gputechconf.com/gtc/2012/presentations/S0096-Summed-Area-Ripmaps.pdf>
(Note for implementation: US patent filed!)

- Connected Components using full GPU parallelism:

<http://on-demand.gputechconf.com/gtc/2013/presentations/S3193-Connected-Components-Kepler.pdf>

geofront

VISUAL COMPUTING

geofront is an Austrian R&D company for computer vision and general purpose computation on graphics hardware (aka GPUs).

Its founder, Dr. Gernot Ziegler, has more than ten years of graphics hardware experience, and is supported by a team of contracted engineers who are dedicated to taking on GPU software

Thank you.
Questions? <http://www.geofront.eu>

We develop novel cutting-edge algorithms, applying real-time graphics concepts to computer vision tasks and other spatial signal processing, such as light wavefront simulation, video compression and computer tomography.

We supply contracted research and software development, leading engineers to implemented solutions. Services include task assessment and white board advice on solution approaches in workshops. These can also be combined with training for GPU development.

Finally, our on-demand advice supports your engineering staff during their graphics

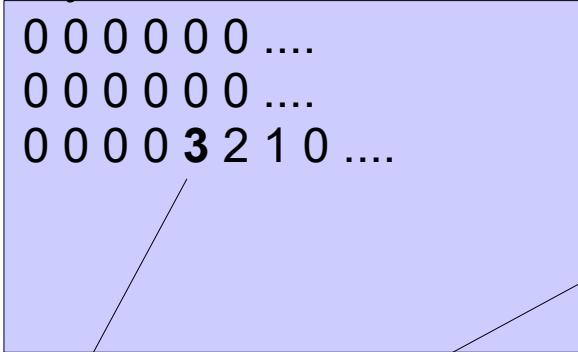
Data Compaction

(the „postponed“
algorithm explanation
from Line List generation)

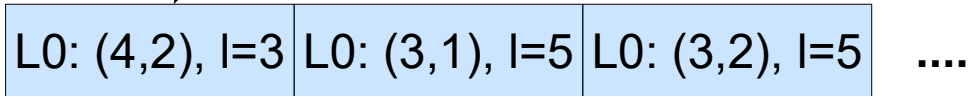
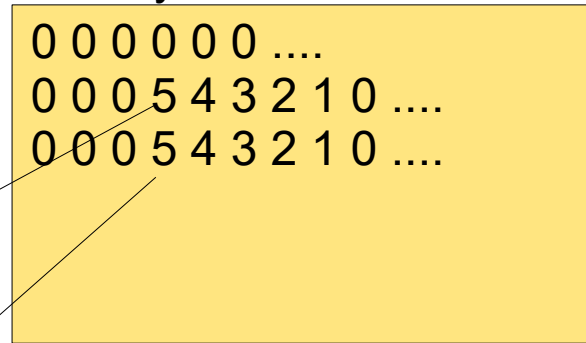
Line Segment Map to Line List

- Inference stores lengths in Line Segment Map
- Create Line list (*) via „Values above Threshold T, preceded by a Zero“

Layer 0: -12.5° to 12.5°



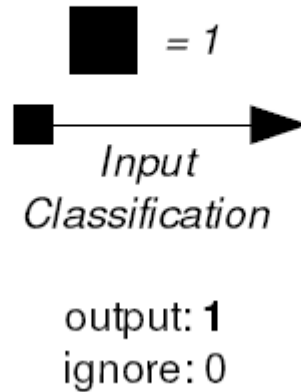
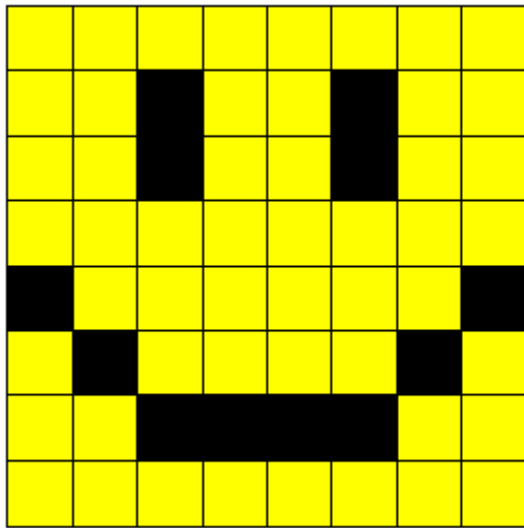
Layer 5: 60° to 80°



- (*) OpenGL or CUDA : Atomics (OpenGL ES, WebGL: Data Compaction, postponed)

Data-Parallel Challenges: Data Compaction

- Our task: Select elements from a larger array, write into a list
- Example from Computer Vision: **List of all black pixels in an image**
- Step 1: Detect black pixels:

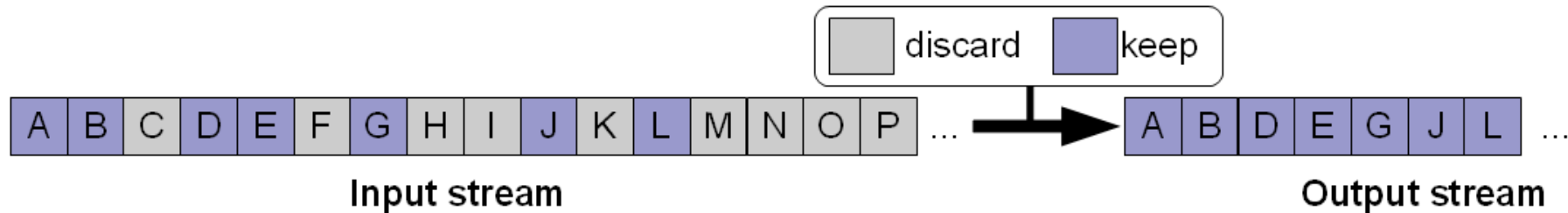


0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0

- Step 2: Create a list of detected pixels

Data Compaction: Problem task in 1D

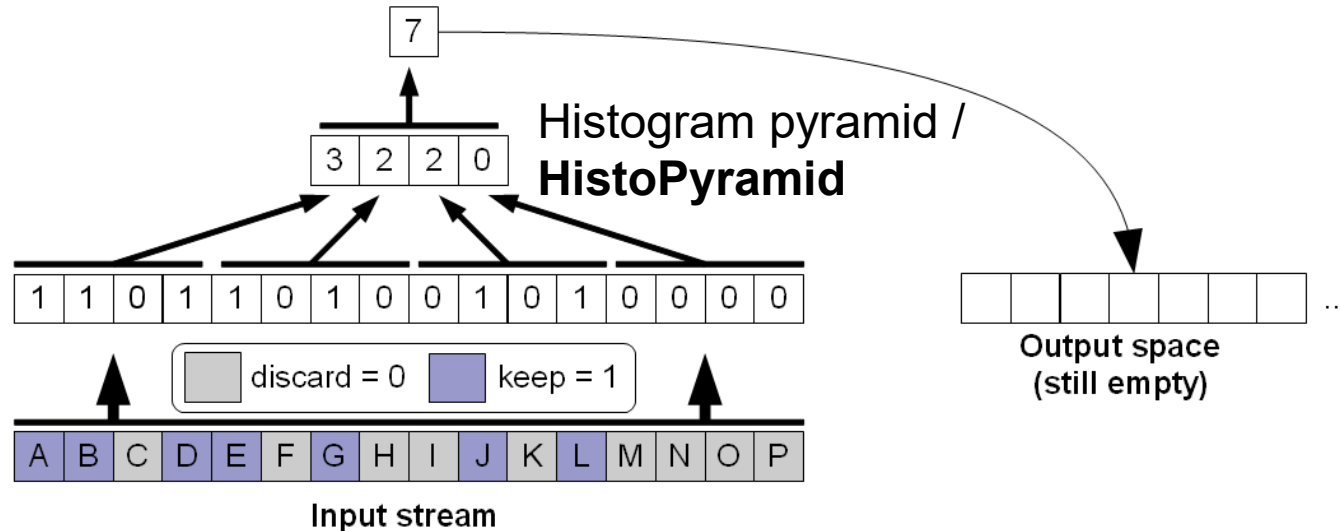
- Keep number of elements from input, based on a *Classifier*:



- Implementation is trivial on CPU, single-thread.
- On GPU: Need to parallelize into 10k threads!
- First *count* number of output elements using data-parallel reduction!

Data Compaction via HistoPyramid:Buildup

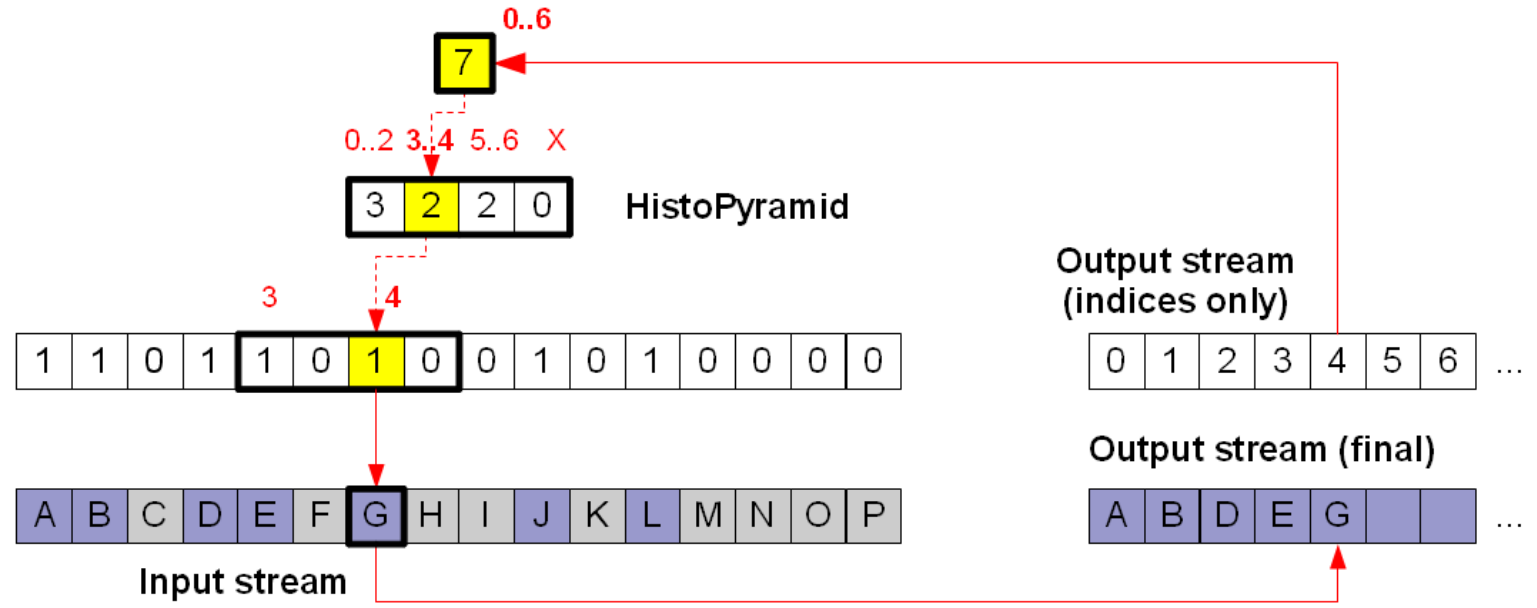
- First, count number of output elements, e.g. 4:1 data-parallel reduction



- (Note the reduction pyramid, it is retained - HistoPyramid)
- Can now allocate compact output, no spill.
- But how are output elements generated?

Data Compaction via HistoPyramid: Traversal

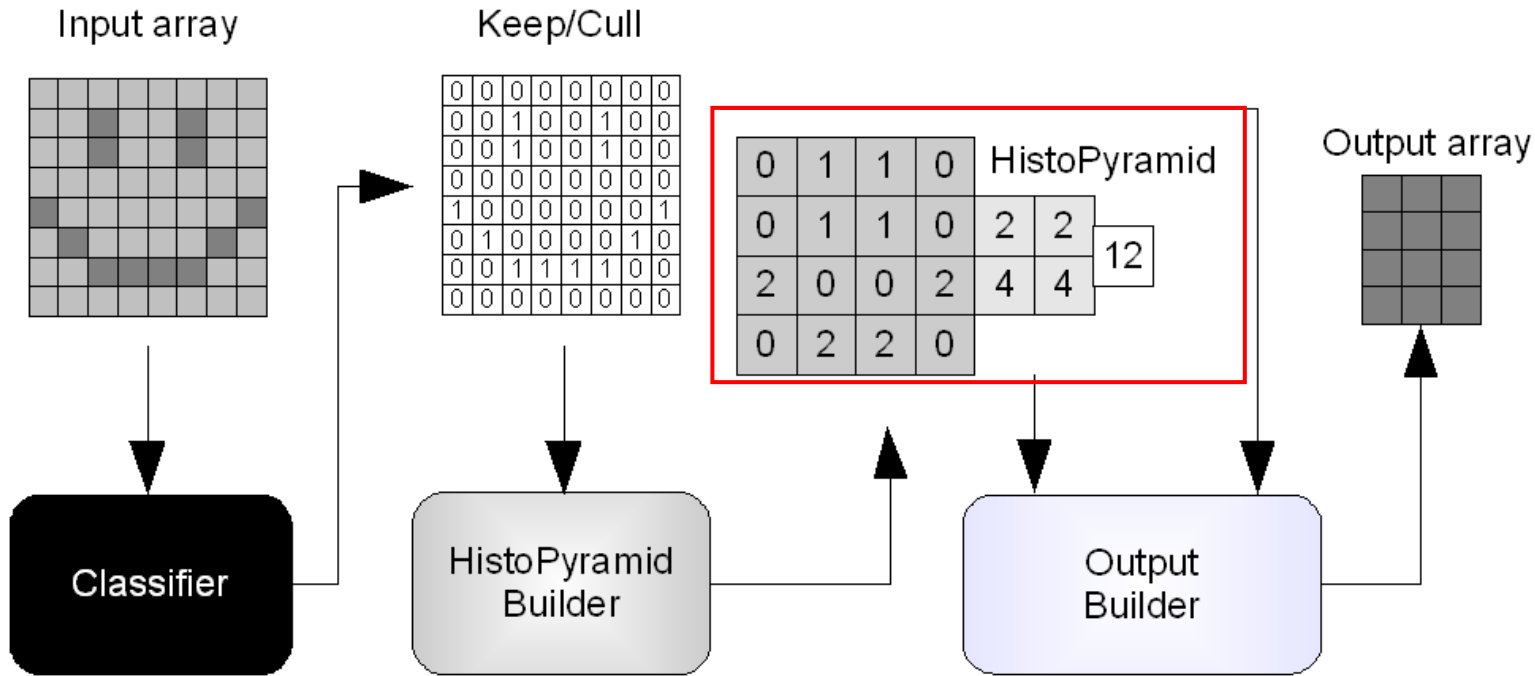
- Output generate: Start one thread per output element
- Each output thread *traverses reduction pyramid (read-only)*



- No read/write hazards = Data-parallel output writing!
- As many threads as output elements

HistoPyramid: 2D Data Compaction

- 1D was tutorial, actual implementation is 2D !
- Dataflow diagram:



Photogrammetry

- Static objects:
Multi-angle acquisition on rotating table, e.g. with automatic camera slider or robotic arm (seriously!)
- Dynamic objects:
Create Multi Camera dome as done by Paul Debevec
<https://www.youtube.com/watch?v=tKCY1dLYBfw#32m00>
(hint: cell phones are cheap now!
And they have their own storage and WLAN...)

Photogrammetry: Future

- Ultimate goal:
Offer acquisition quality as demo:ed by Realities.io,
but here in Vienna:
https://www.youtube.com/watch?v=B8FPunc_RTE
- Reconstruction of Museums, Castles, Architecture, ...
plenty of vision and graphics work
- **Then it's time to buy an HTC Vive -**
for the Austrian tourism agencies around the world. ;)

GeoCast & GeoScene

May 2016

<https://www.geofront.eu/blog/geoscene-and-geocast-formats.html>

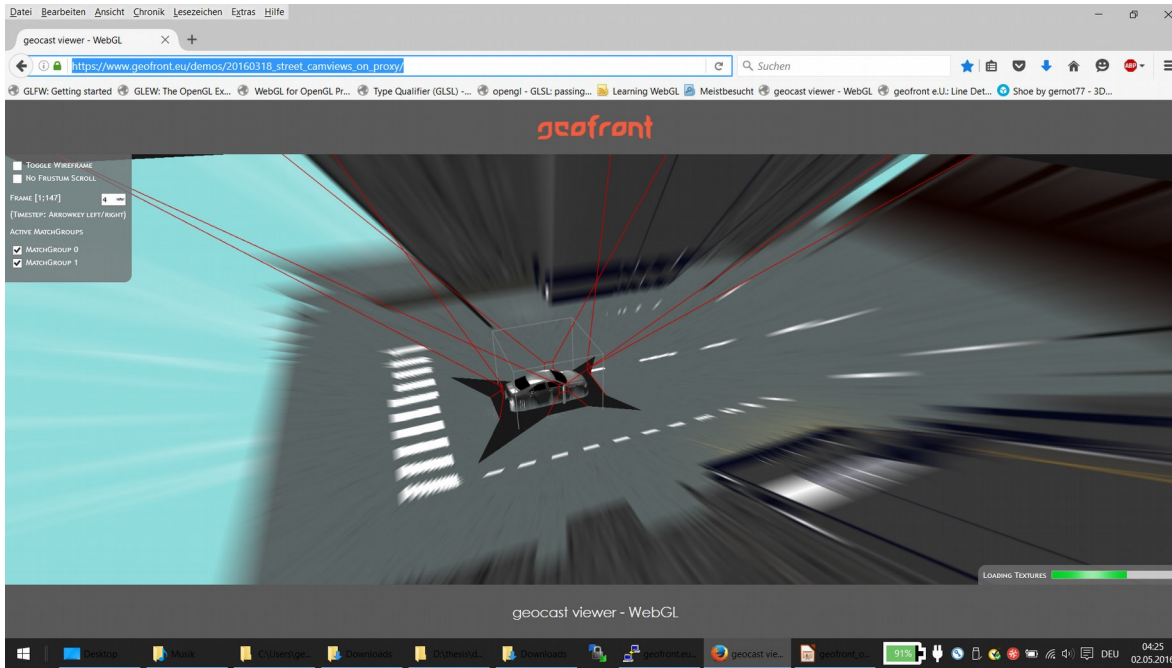
- Data Exchange between Computer Vision and Computer Graphics requires exact viewing ray mapping, also for depth maps
- GeoCast provides common data format using OpenGL!
- Useful for virtual simulation testing of computer vision algorithms (e.g. AIT has started using Blender for this)
- Multi-view scene reconstruction requires complete camera paths as well.
- Generate arbitrary testing scenarios, and ground truth.

Other Projects

GeoCast & GeoScene

Top view and more using proxy geometry

May 2016



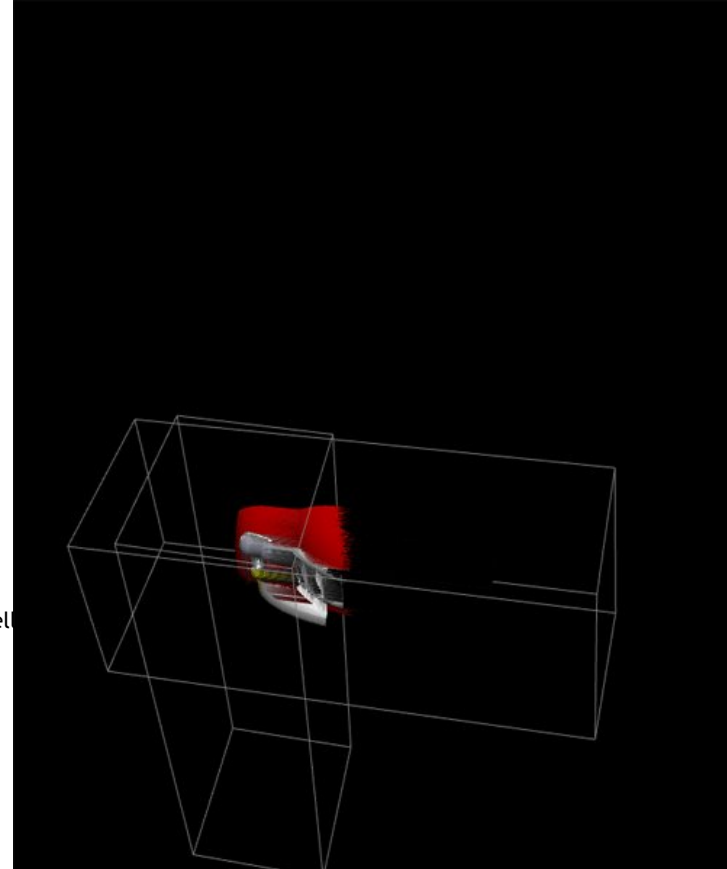
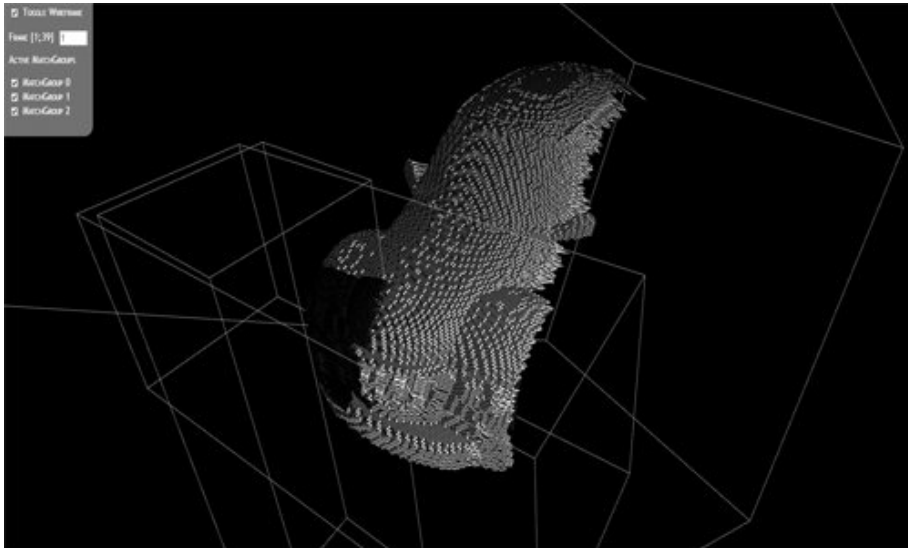
- With proxy geometry such as a quad describing the street level, the GPU can reconstruct car viewpoints or help debug vision and decision making of car AI.
- https://www.geofront.eu/demos/20160318_street_camviews_on_proxy/

geofront
Fast Line Segment Detection

39 of 41

GeoCast & GeoScene „Depth map projection“

May 2016



- GeoCast facilitates data exchange with data sources such as depth cameras, laser scanners, 3d model
- GPU helps reconstructs novel views from partial depth surfaces using Z-buffer

geofront
Fast Line Segment Detection

40 of 41

Comparison to other approaches

- Hough transform

Can in theory detect lines at any angle, but

- (a1) massive bandwidth for evaluation of all angle hypotheses (similar to FFT convolution), OR
- (a2) relies on incomplete hypotheses verification (undersampled)
- (b) requires analysis of transform result,
- (c) cannot detect line segment position and length!