Eurographics Conference 2010
FP16 Rendering II

SCALABLE HEIGHT FIELD SELF-SHADOWING

Ville Timonen[1,2]
Jan Westerholm[1]

[1]Åbo Akademi University
[2]Turku Center for Computer Science

PRESENTER NOTES

CONTENTS

3 - Core concept
4 - Slightly different computation scheme, not doable w/ current shaders

I PROBLEM DESCRIPTION

I PROBLEM DESCRIPTION

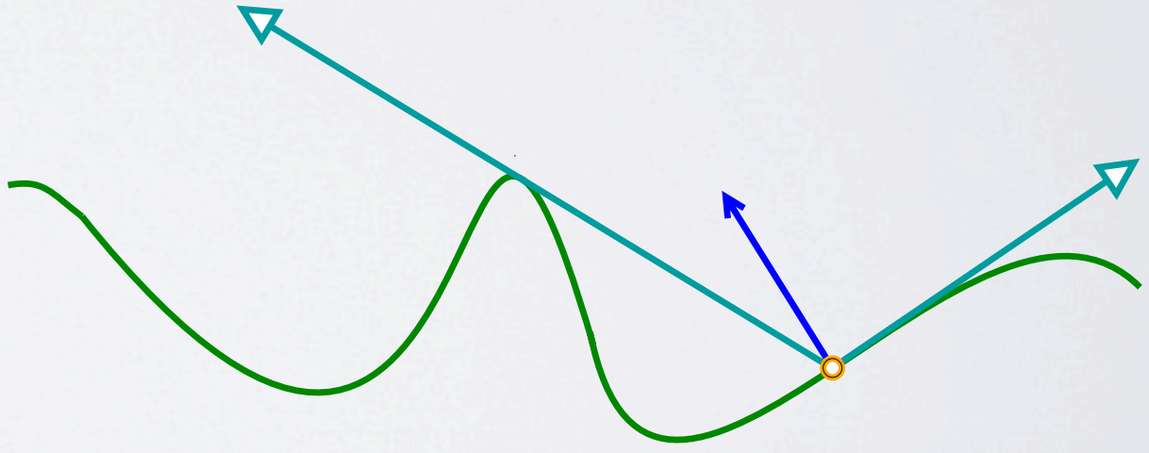
Height field geometry

Specific geometry. A representation of a height field on the right, displaced surface in the back.
Elevation of a surface is defined as a function (HF) of discrete surface coordinates.
HFs can be used to e.g. introduce microgeometry to crude polygon models or to represent complete objects by themselves

I PROBLEM DESCRIPTION

Direct lighting of height field geometry from an infinitely distant arbitrary light source...

...consists of integrating the input light over the sky's visibility



Sky's visibility <=> surface self-occlusion
Integration over visibility: incoming color * dot(incoming light vector, normal)

1 PROBLEM DESCRIPTION

The two core problems:

1 Integral bounds: visibility of the sky
(Primary contribution, focus of this presentation)

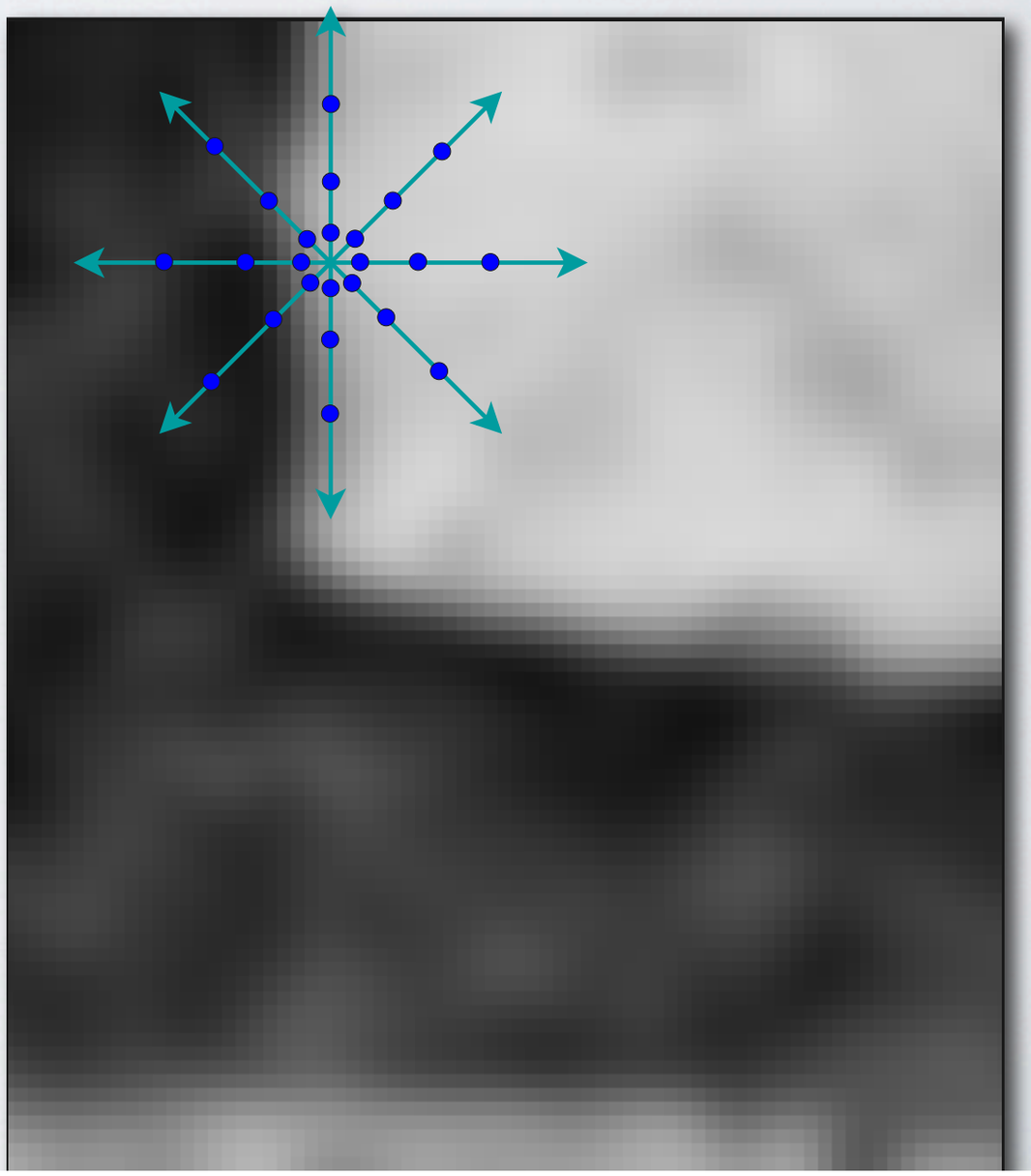
2 Integration of incident light over visibility
(Secondary contribution)

Problem splits down into 2 subproblems. We offer contributions to both but focus on 1
1 Limits of the sky from the perspective of each point in the height field
2 This phase is lighting

I PROBLEM DESCRIPTION

The real challenge is finding the integral bounds

Find horizon as a function of azim. direction for each point in the HF



Naïvely $O(N^3)$ for one direction on an NxN HF

We can solve the visibility by finding the horizon in discrete directions

2 PREVIOUS APPROACHES

2 PREVIOUS APPROACHES

Sophisticated interpolation between different resolutions

Eurographics Symposium on Rendering (2008)
Steve Marschner and Michael Wimmer (Guest Editors)

Fast Soft Self-Shadowing on Dynamic Height Fields

John Snyder[1] and Derek Nowrouzezahrai[2]

[1]Microsoft Research
[2]Dynamic Graphics Project, University of Toronto

Abstract

We present a new, real-time method for rendering soft shadows from large light sources or lighting environments on dynamic height fields. The method first computes a horizon map for a set of azimuthal directions. To reduce sampling, we compute a multi-resolution pyramid on the height field. Coarser pyramid levels are indexed as the

When sampling farther from the receiver, use lower resolution height field
This also uses 4th order SH for lighting, only suitable for soft shadows.
(We use an accurate lighting model)

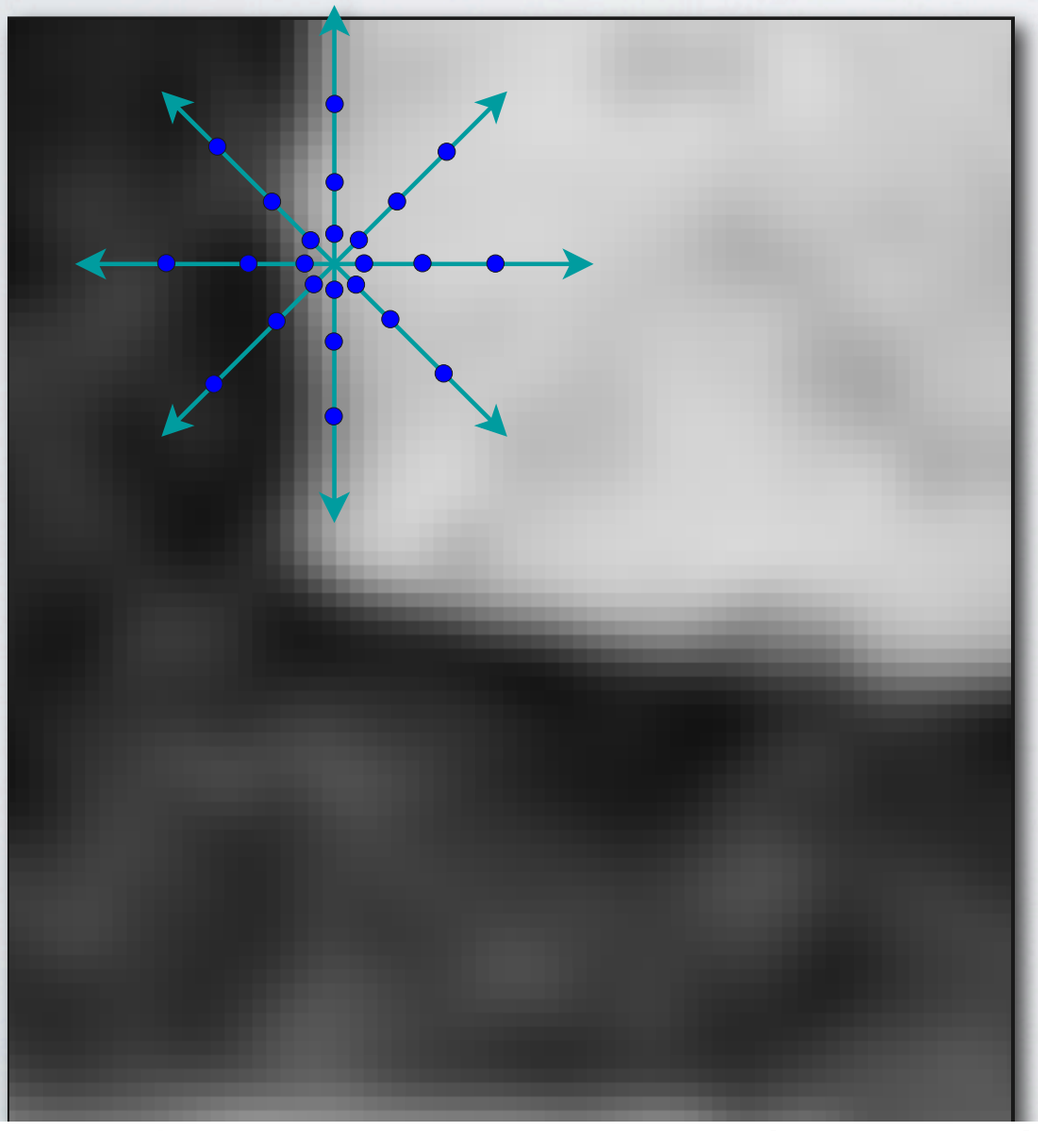
2 PREVIOUS APPROACHES

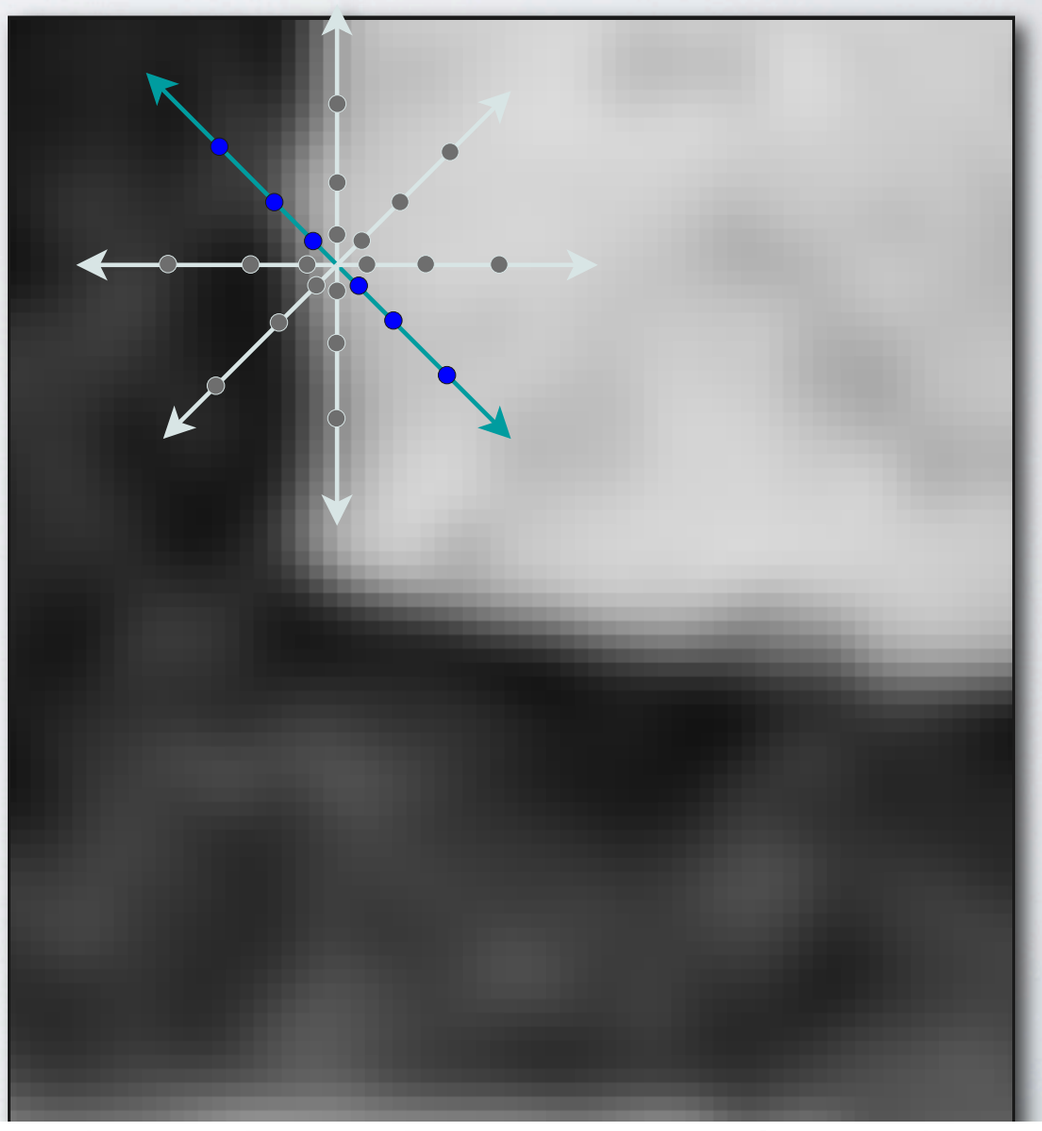
Screen space ambient occlusion: local sampling and artefact hiding

Depth buffer is used as the height field
Occluders are searched locally, as distant samples are unlikely to represent continuous geometry
Randomized sampling directions to trade banding for noise. E.g. bilateral blur for hiding noise

2 PREVIOUS APPROACHES

Room for improvement: fragments on a line undergo redundant processing



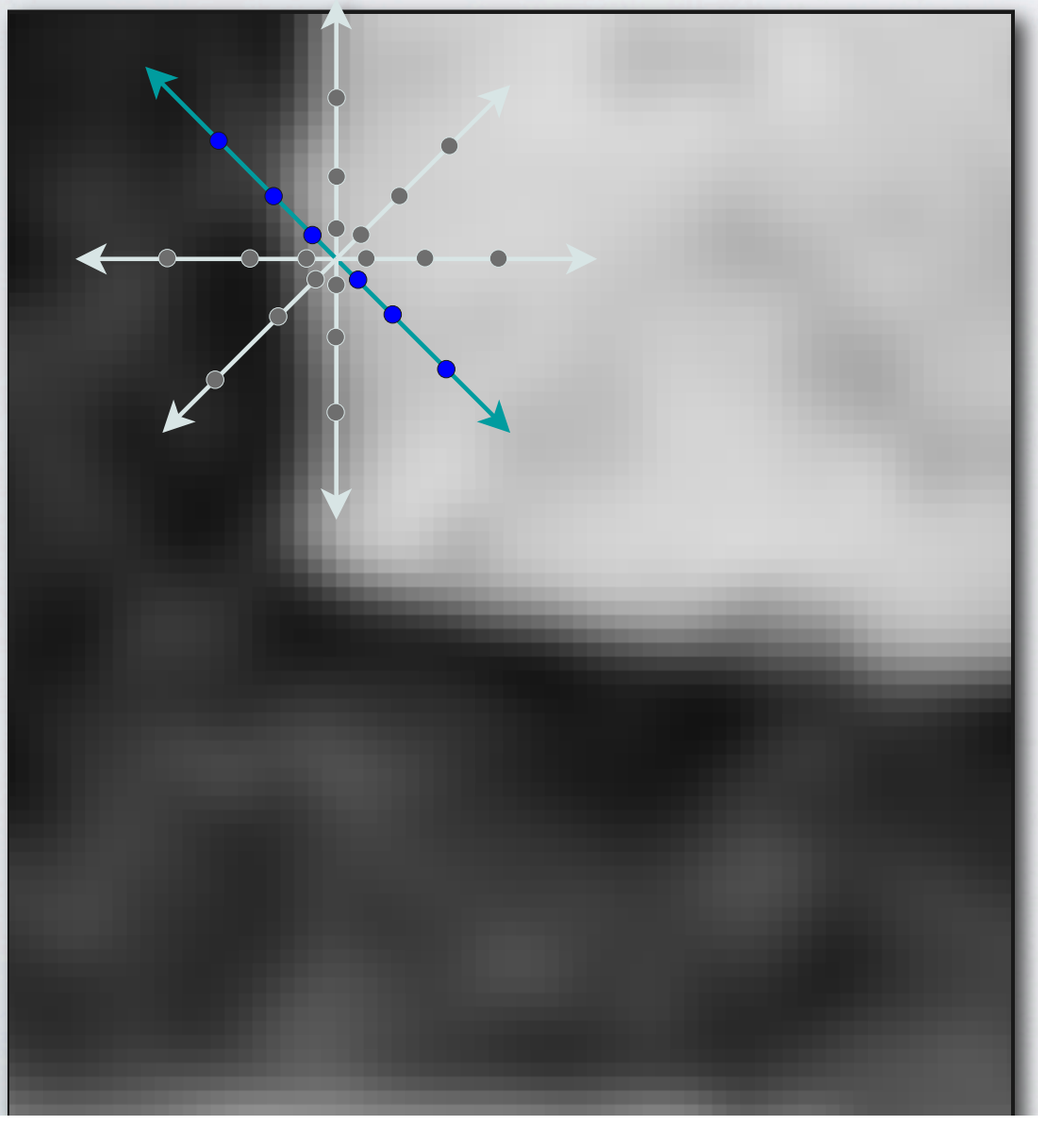


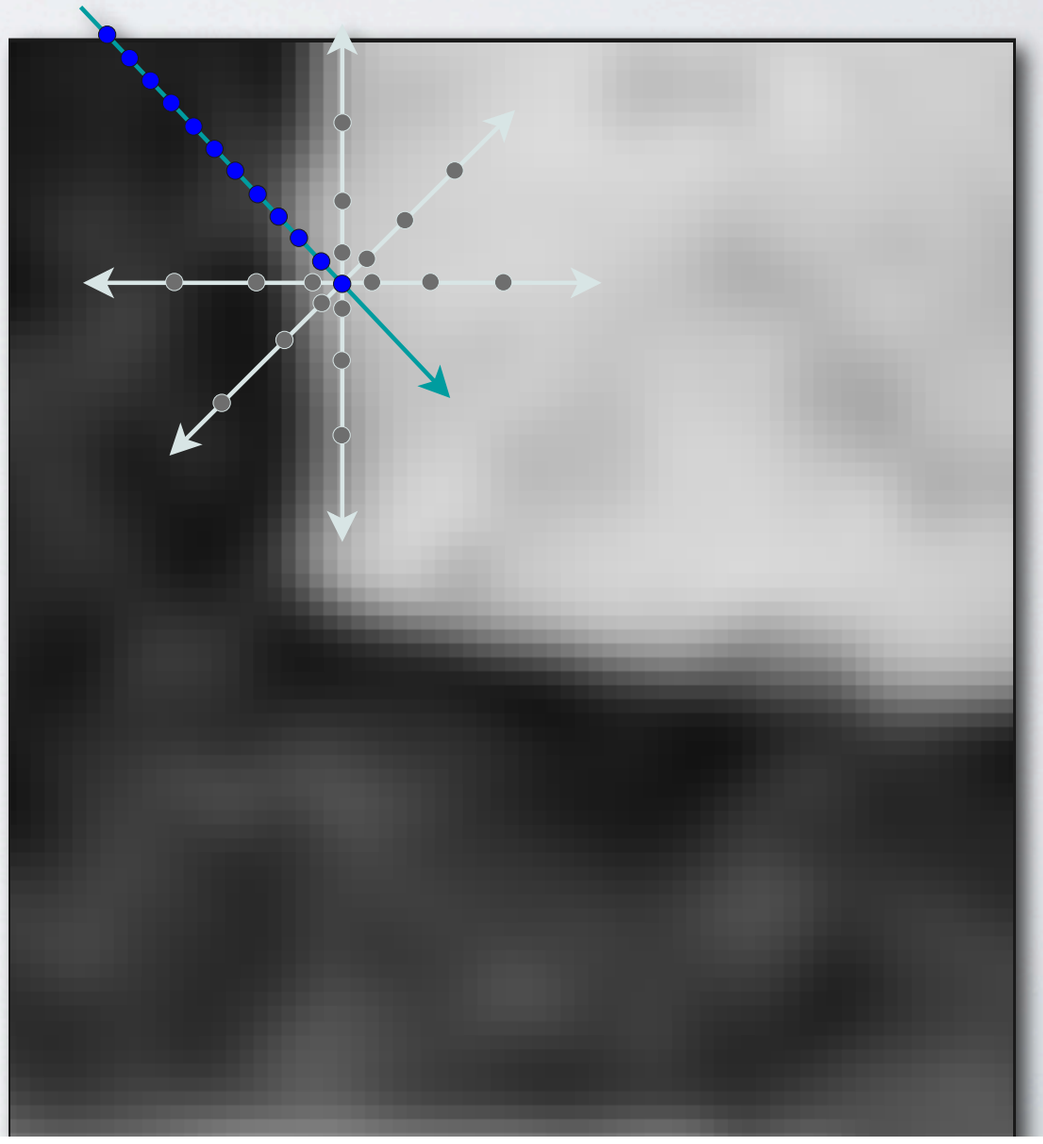
If you consider processing points along one line, each go through same samples. This is redundant processing, at the very least redundant sampling. Our method is based on this premise

3 OUR APPROACH

3 OUR APPROACH

We can traverse through the height field
in lines, instead of independent points

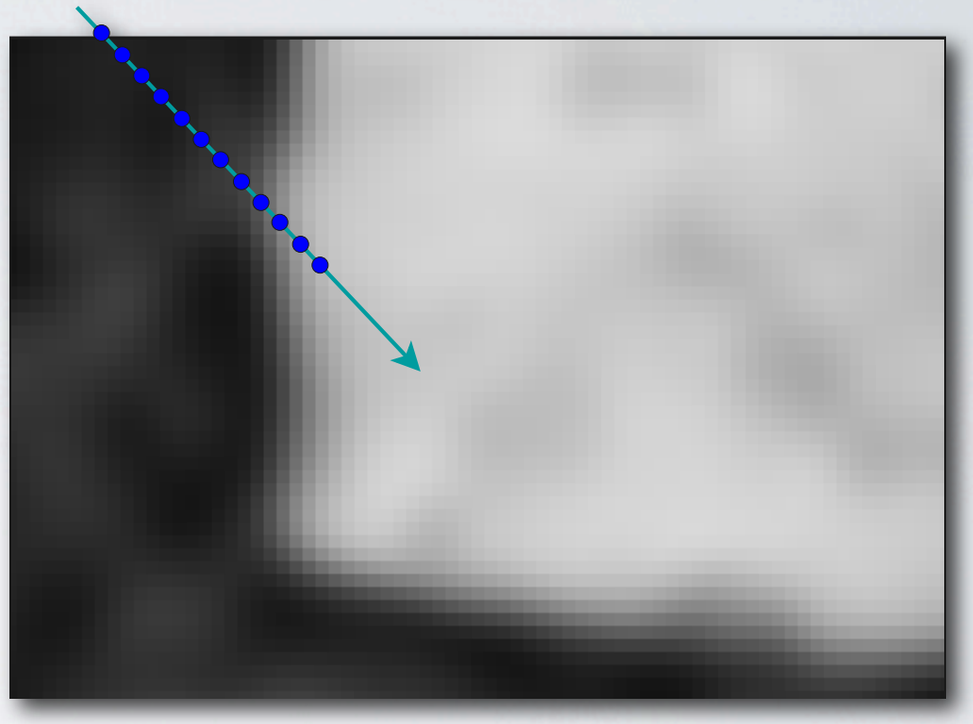




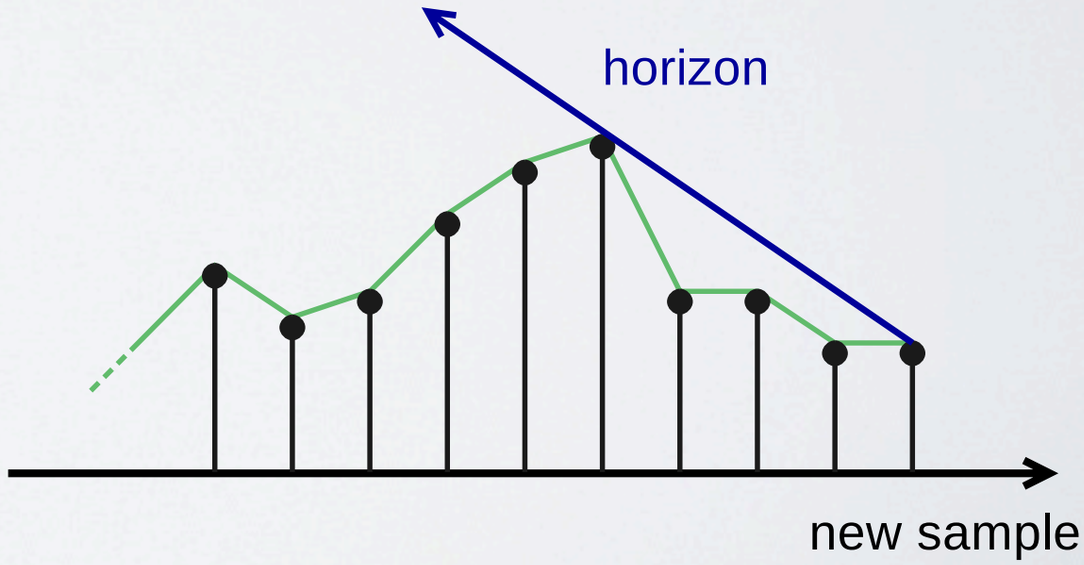
Idea is to process whole lines, so that we can utilize whatever coherency there is in processing adjacent samples on a line

3 OUR APPROACH

When marching along the line, previous height values are remembered



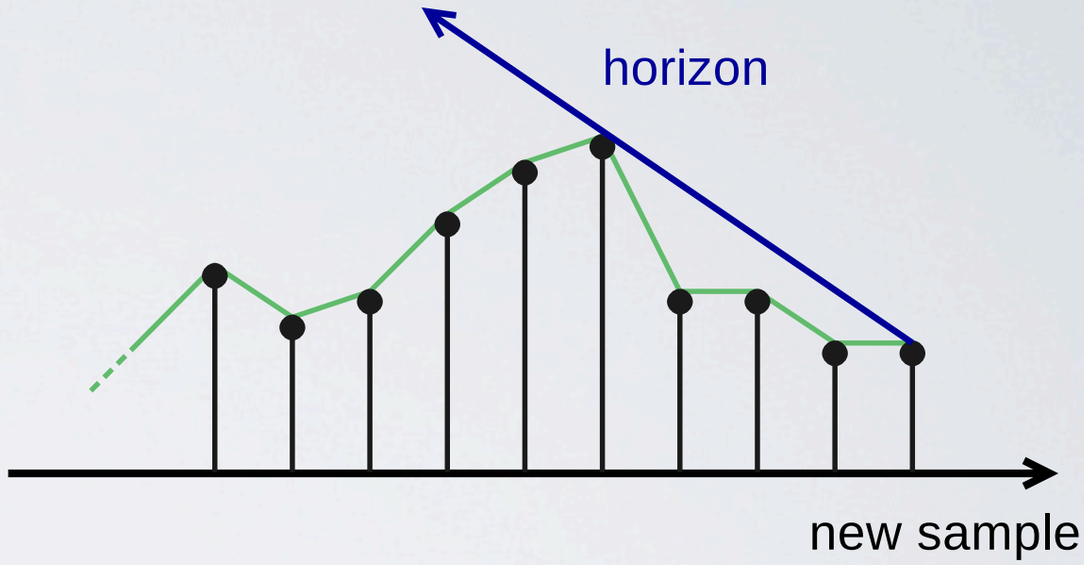
Horizon values are extracted from this internal representation

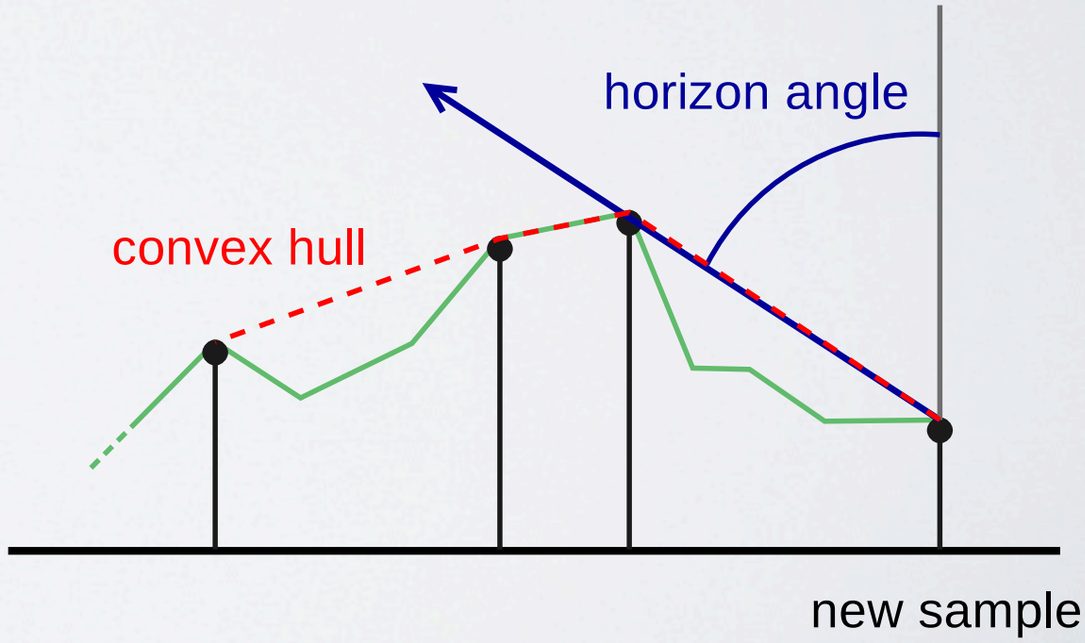


We march one step at a time, and use an internal representation of the height function so far

3 OUR APPROACH

Maintain a **convex hull subset** of all samples so far, and the horizon angle for each new sample is determined by the previous value in the subset

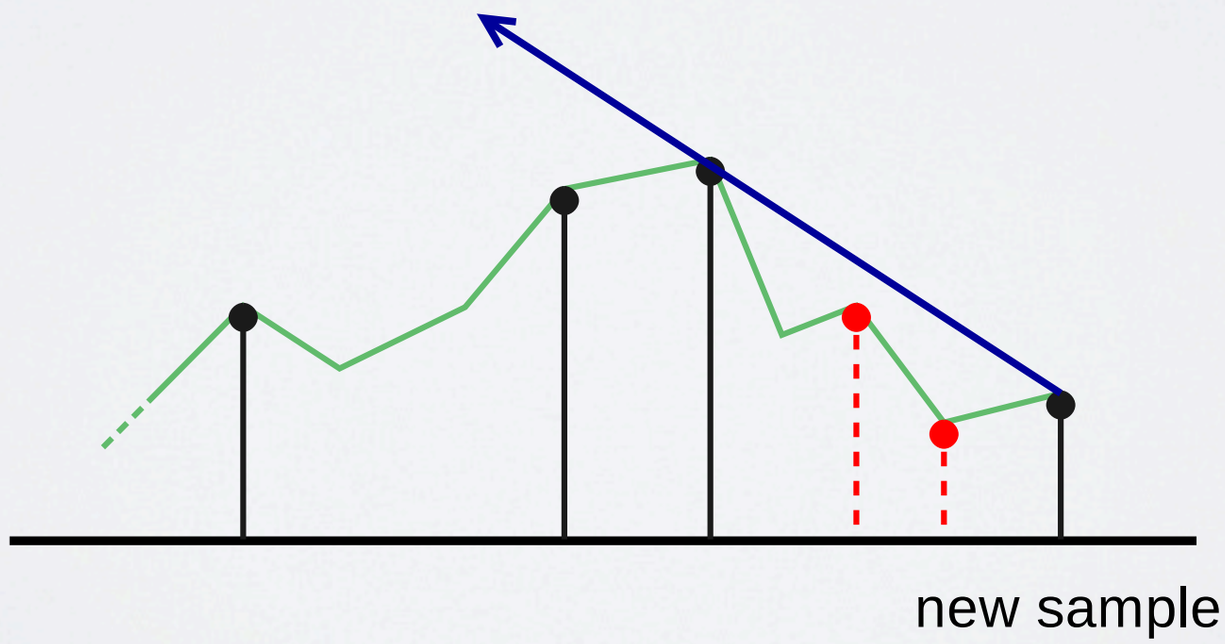




Now this is important:
If we consider the convex hull subset of all samples, including the new sample, the horizon angle for the new sample cannot be determined by any other sample than one of the convex hull subset. Furthermore, each element in the set can have a direct line of sight only to its neighbors, so the horizon angle backwards is determined by the previous element in the set

3 OUR APPROACH

All we need to do when processing the next sample is to add it to the subset while retaining convexity

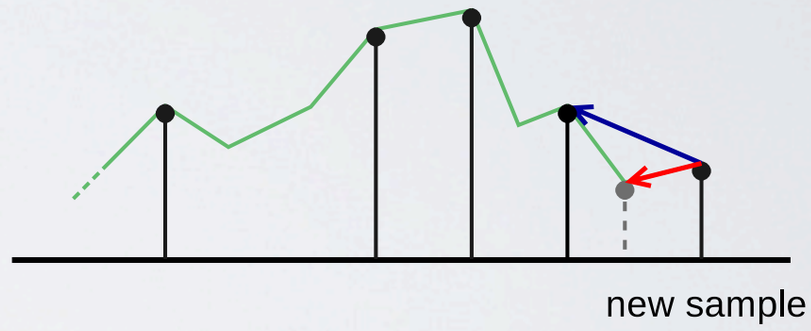


In this case for example, the previous elements in the set (marked in red), are to be removed

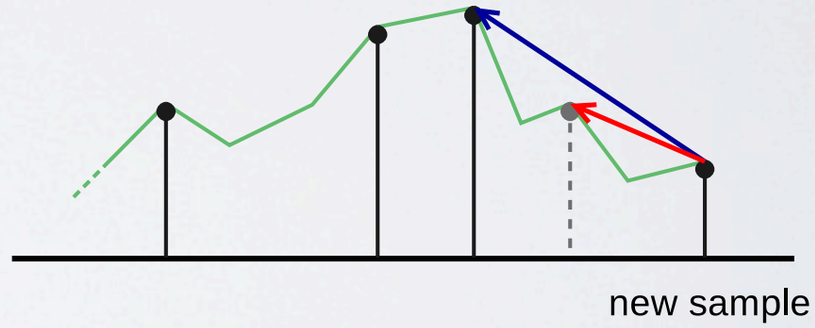
3 OUR APPROACH

Store the convex hull subset in a stack, pop it until the 2 topmost values and the new sample are convex

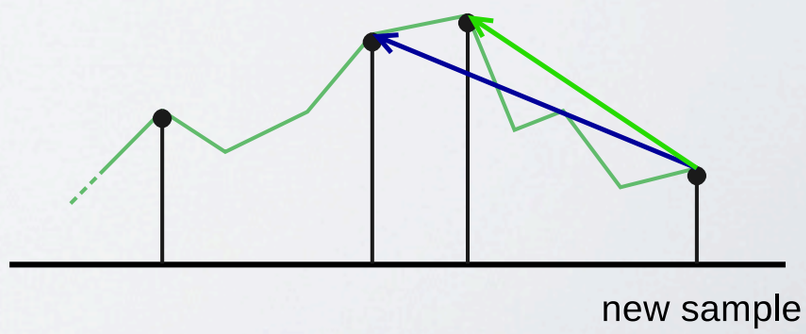
1st iteration fail → pop



2nd iteration fail → pop



3rd iteration pass → push(new)



Each time you add a sample, you pop the stack until the tail is convex
You can check the convexity easily by comparing vector slopes

3 OUR APPROACH

A line of N samples:

N pushes to the stack

<N pops (one iter. each)

N iterations without a pop

At most 2N iterations total

Algorithm 1 Processing a new height map sample *new*

$v_1 \leftarrow \text{VECTOR}(peek_1 \rightarrow new)$
while $size > 1$ **do**
  $v_2 \leftarrow \text{VECTOR}(peek_2 \rightarrow new)$
  if $h(v_2)d(v_1) \geq h(v_1)d(v_2)$ **then**
    $v_1 \leftarrow v_2$
    pop
  else
    break
  end if
end while
push(*new*)
return $\frac{\pi}{2} - tan^{-1}\frac{h(v_1)}{d(v_1)}$

Computational complexity is O(N)

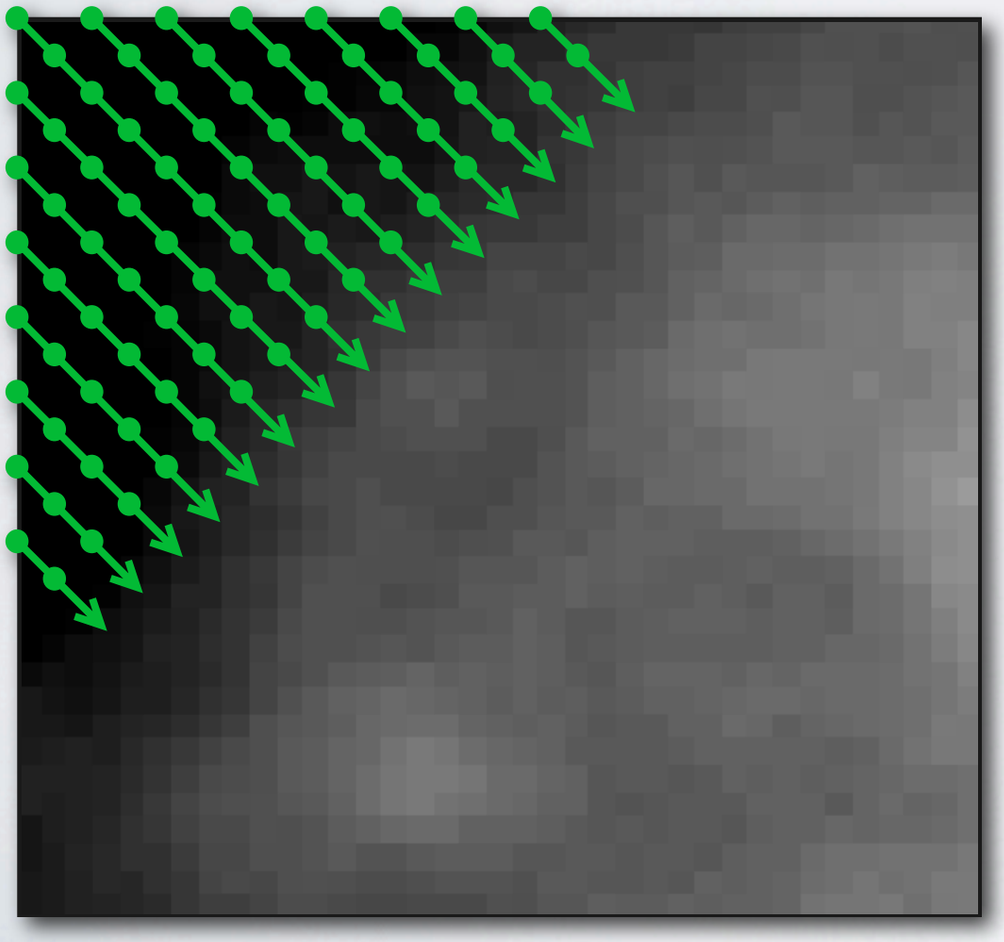
The algorithm is also very compact and fast to execute on GPUs

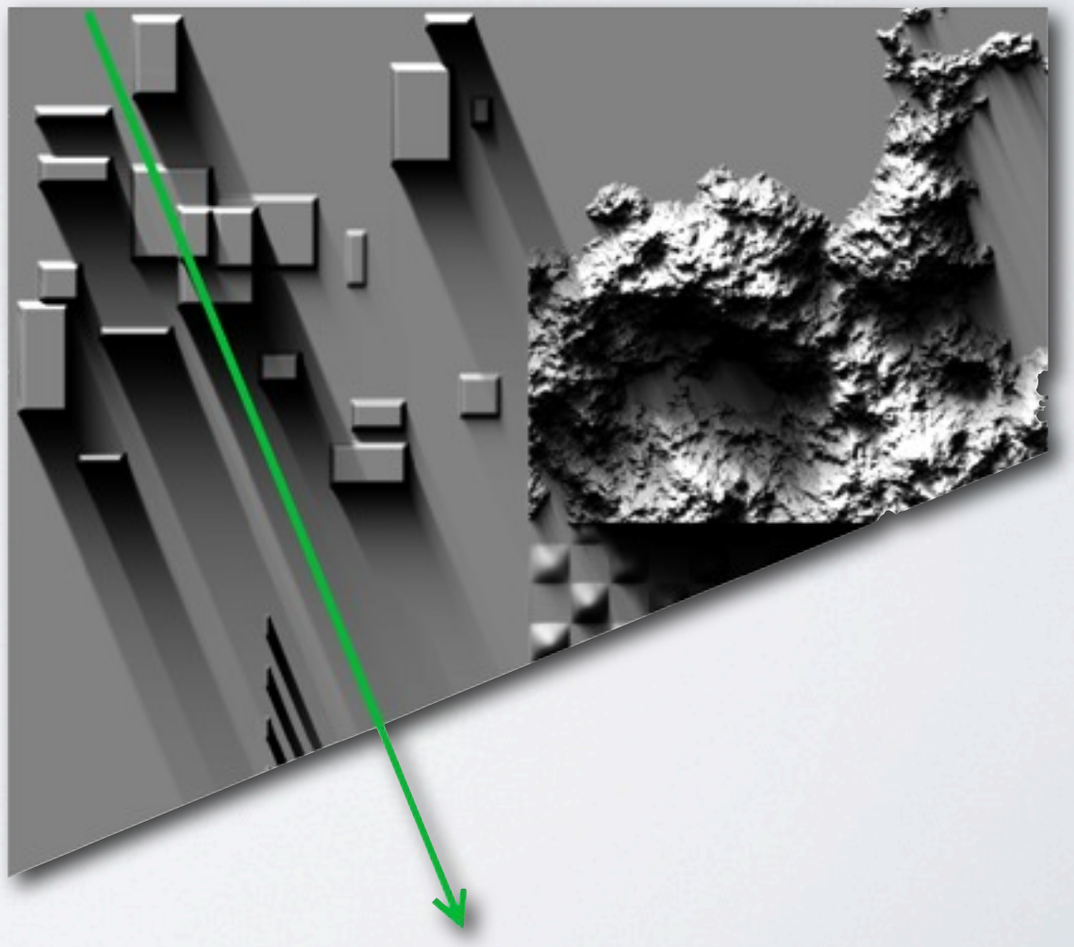
4 COMPUTATION FRAMEWORK

4 COMPUTATION FRAMEWORK

Processing along parallel lines

Not a suitable candidate for a fragment program





We process lines, not pixels as with fragment programs, so we need to write multiple locations across an output buffer from a thread.

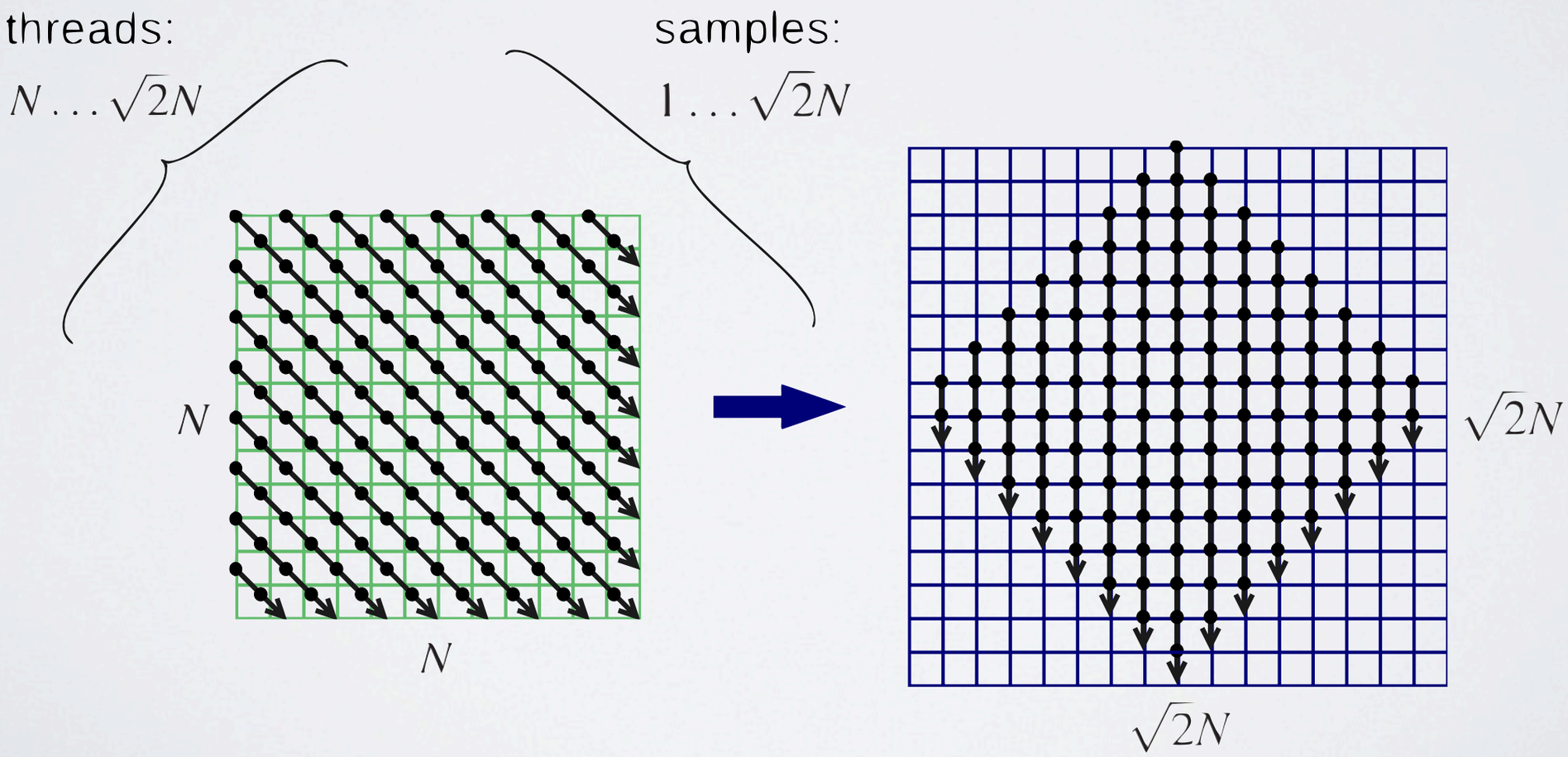
Scatter writing not possible with fragment shaders

4 COMPUTATION FRAMEWORK

Calls for GPGPU, we use CUDA

Memory coalescing still possible

Enough threads to utilize a GPU



However, this still maps efficiently to graphics hardware.
(1) We can still utilize memory coalescing by writing results to an aligned output buffer (GPU mem. ctrller only requires 1D write patterns)
(2) We can have enough threads to utilize a GPU, especially when we process multiple directions at once (89k threads for 1024*1024 HF w/ 64 dirs) -- trivially parallelized
We can build thread blocks so that thread spans don't differ that much but memory coalescing is still possible

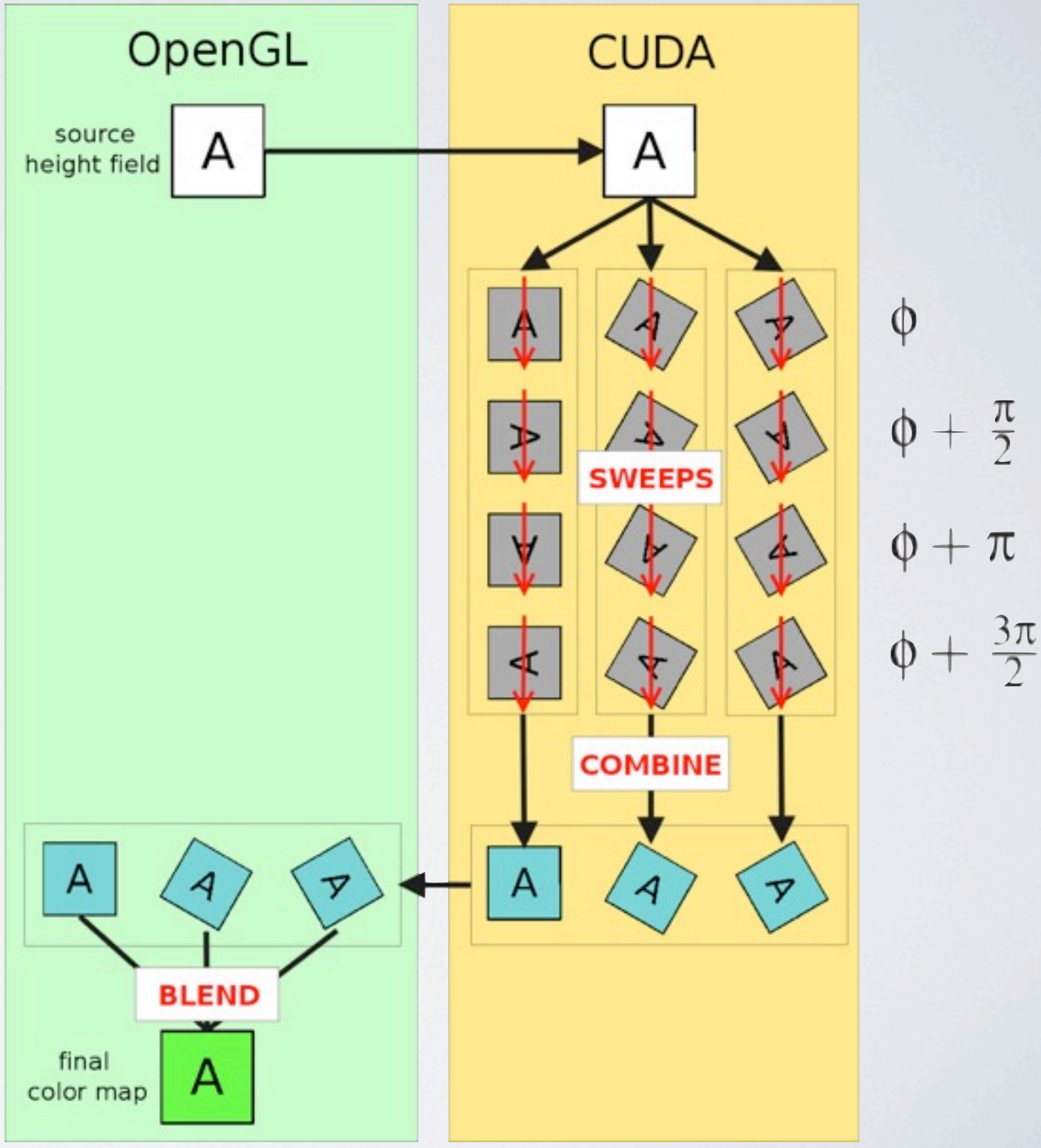
4 COMPUTATION FRAMEWORK

OpenGL PBO to CUDA Array

One sweep for each direction
(in this example 0°, 30°, 60° ... 330°)

90° rotations are linearly combined for reduced overhead in OGL interop.

Resulting textures are blended in OpenGL

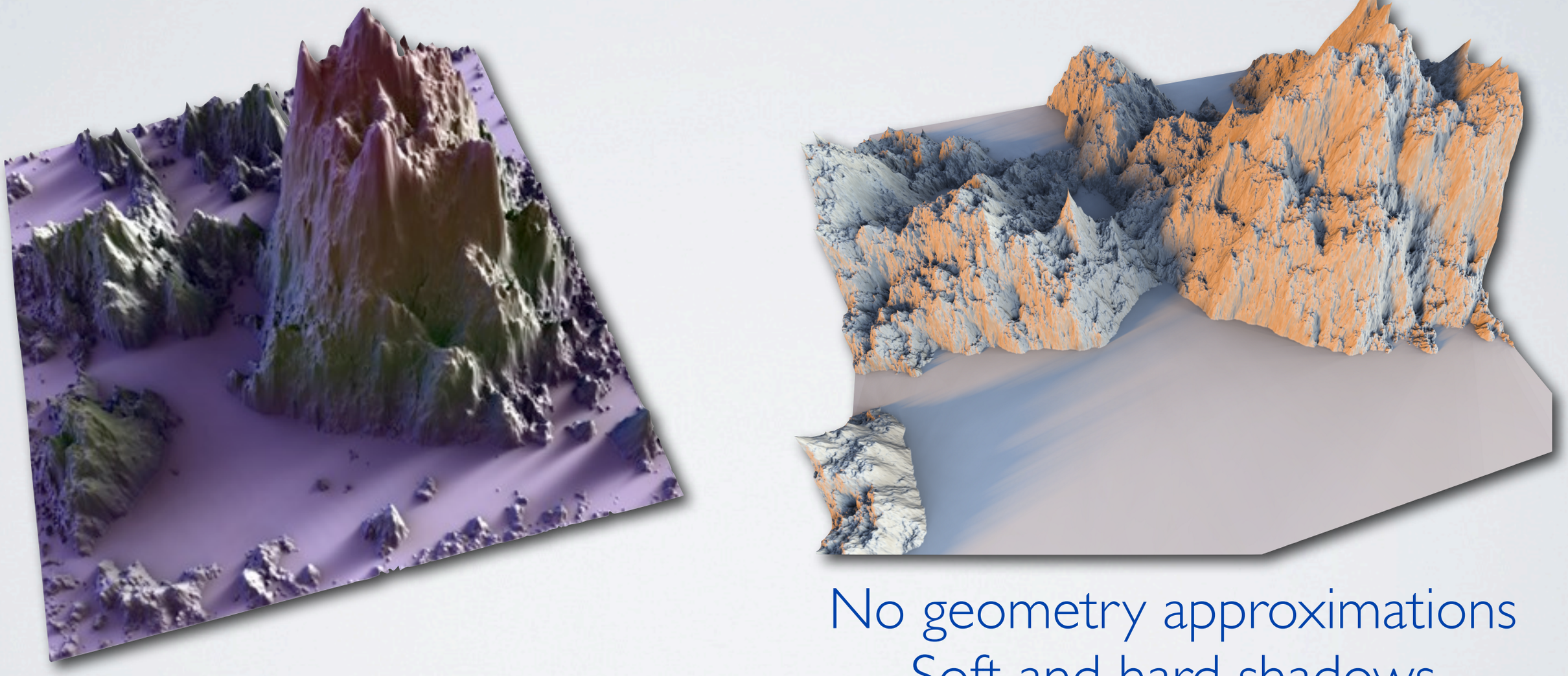


As many sweeps (each generating horizon angles for one direction) as possible is performed at once, limited by how much VRAM can be used
During combining, also lighting is calculated, but not included in this presentation
There's expensive overhead in current OGL interop (APIs & their implementations rapidly changing, though) so we combine some textures already in CUDA. The whole pipeline uses HDR values

5 RESULTS

5 RESULTS

Time complexity reduction from $O(N^3)$ to $O(N^2)$



No geometry approximations
Soft and hard shadows

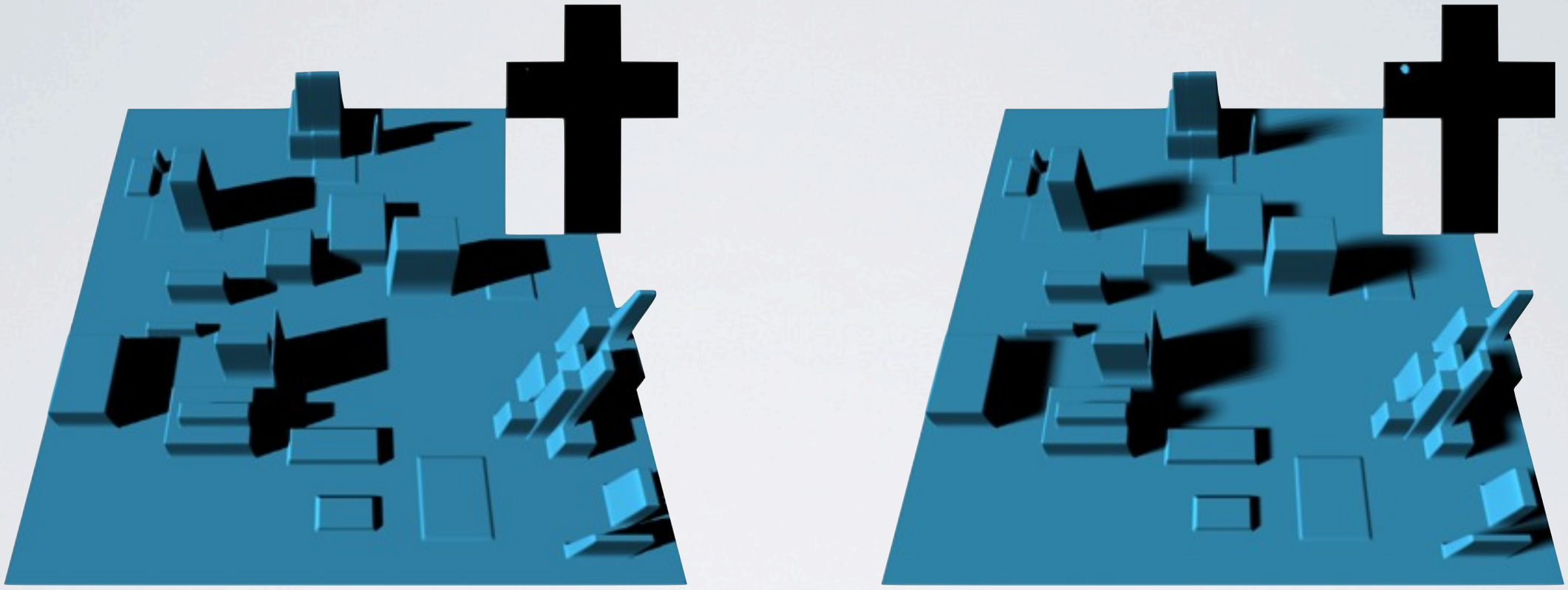
2.5 Hz
Snyder et al.

1024^2 HF, 16 azim. dirs

38 Hz
Our method

To the left is the previous state-of-the-art from Snyder et al.
This is 1024^2 height field shadowed for 16 azimuthal directions on an G80 card (same card for both)
We are able to do both soft and hard shadows, because we don't approximate geometry (and use an accurate lighting model). And we're faster too, even more so when HF size, lighting resolution and azim. dir. count grows.

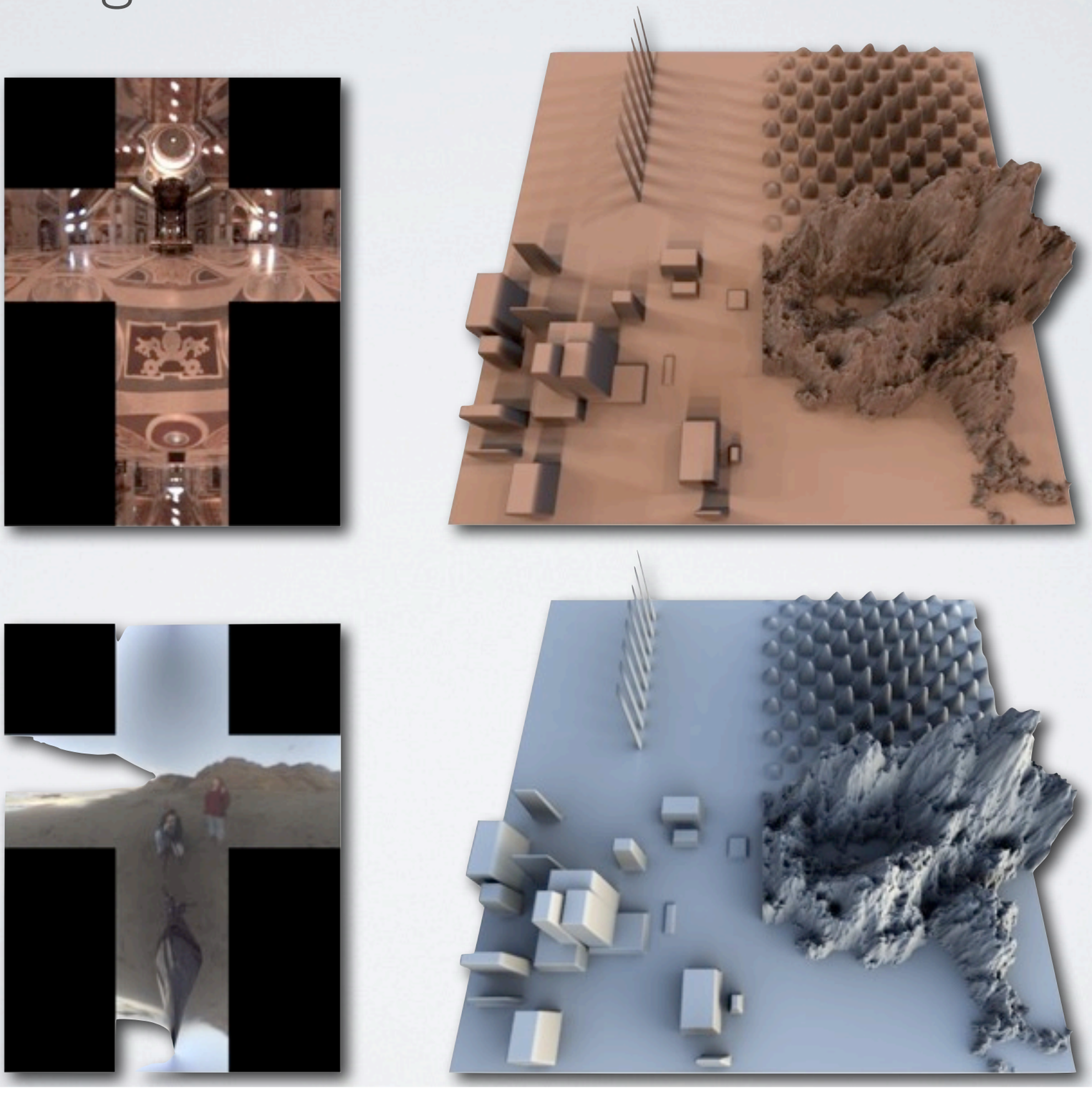
5 RESULTS



Here you can see this method’s ability to handle both hard and soft shadows

5 RESULTS

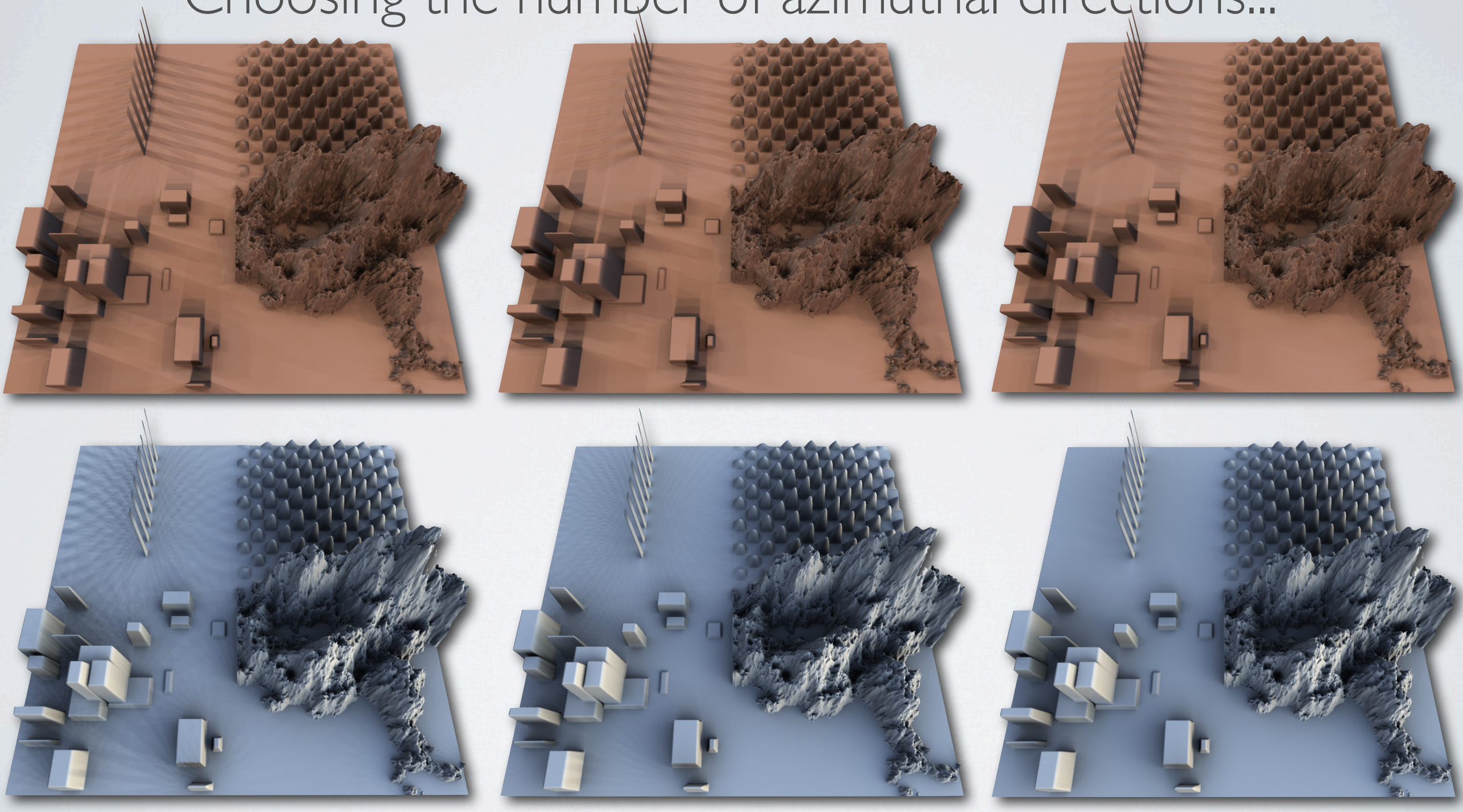
Choosing the number of azimuthal directions...



HDR cube map lighting environments on the left, 1024^2 height maps on the right
Let’s see how azimuthal directions affect

5 RESULTS

Choosing the number of azimuthal directions...



16 32 64

Here you can see the effect of the number of azimuthal directions. For worst cases, even 128 might be required to acquire smoothness. Then again, 16 might be enough for complex geometry where shadows don't produce that visible boundaries

5 RESULTS

Performance figures

HF res.	FPS for uniform, environment lighting				
	$\phi_N = 16$	32	64	128	256
	Nvidia GTX 280 (1 GB)				
512^2	160, 114	118, 87	74, 61	44, 38	24, 22
1024^2	76, 62	45, 39	24, 23	12, 12	6.4, 6.3
2048^2	24, 24	13, 13	6.6, 6.7	3.4, 3.4	1.7, 1.7
4096^2	5.4, 6.8	2.8, 3.5	1.4, 1.8	0.7, 0.9	0.4, 0.5
	Nvidia 8800 GTS (512 MB)				
512^2	110, 108	62, 68	33, 38	17, 20	8.6, 10
1024^2	33, 38	17, 21	8.8, 11	4.4, 5.5	2.2, 2.8
2048^2	8.8, 10	4.6, 5.4	2.3, 2.8	1.2, 1.4	0.6, 0.7

It's pretty much linear in height field size and the number of azimuthal directions
For example 1024^2 with 64 directions is 23 fps. Rather insensitive to geometric content.
If you're interested in some very high definition for e.g. offline graphics, you can do 4096x4096 height fields (34M polygons) with 1024 directions in 8 secs a frame

6 VIDEO DEMONSTRATION

QUESTIONS?