graphics hardware 06

High Quality Normal Map Compression

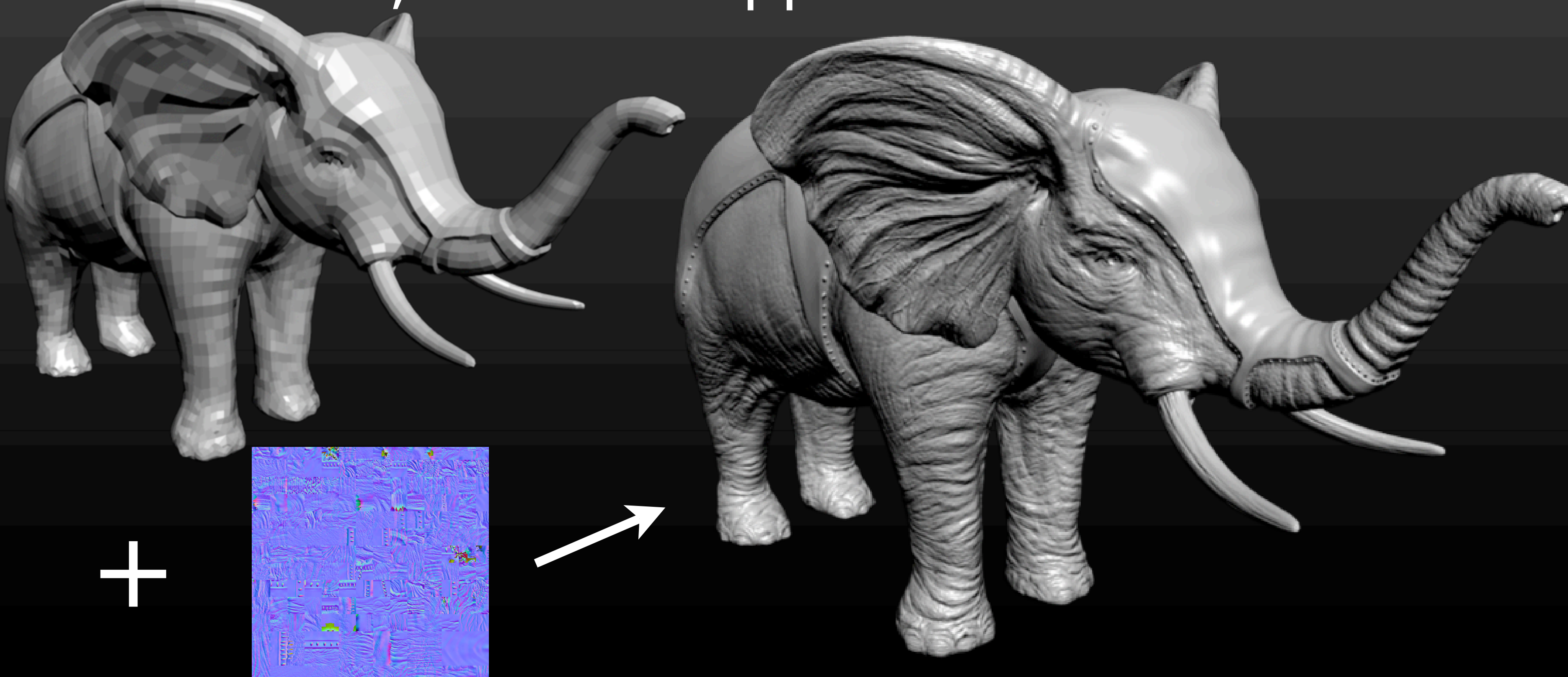
Jacob Munkberg — Lund University

Tomas Akenine-Möller — Lund University

Jacob Ström — Ericsson Research

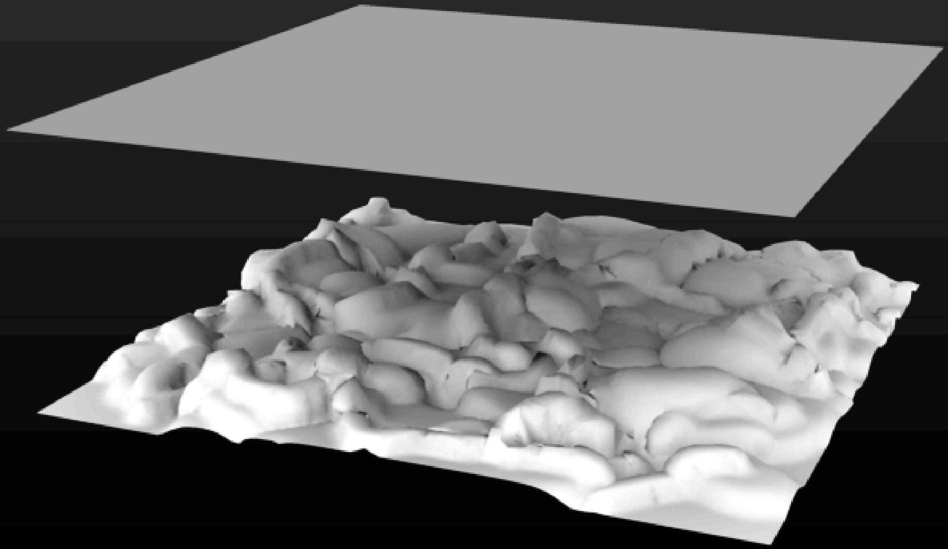
Normal Maps

- Add geometric detail with texture maps
- Store value of the local normal vector
- Realistic, detailed appearance at low cost



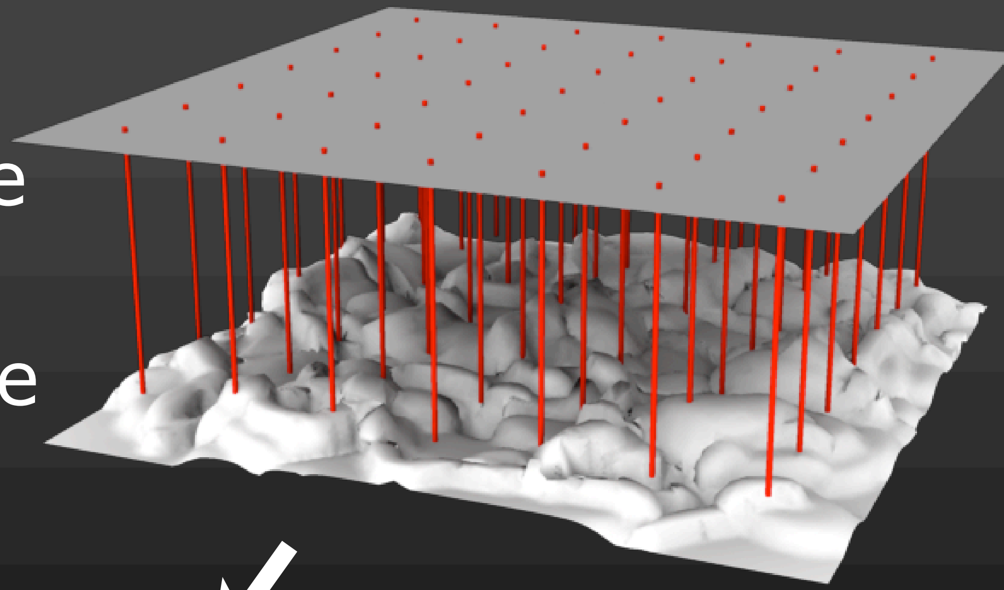
Normal Map Generation

- Create two versions of the mesh
  - Lo-res mesh - overall shape
  - Hi-res mesh - shape + details

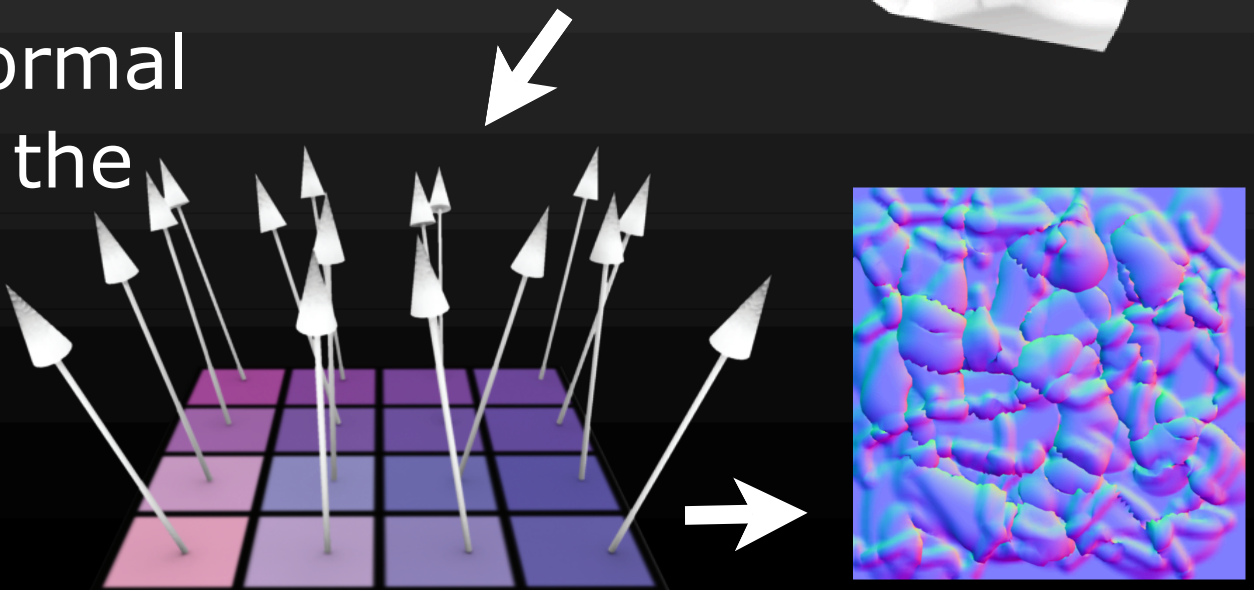


Normal Map Generation

- Shoot rays from the lo-res surface to the hi-res surface

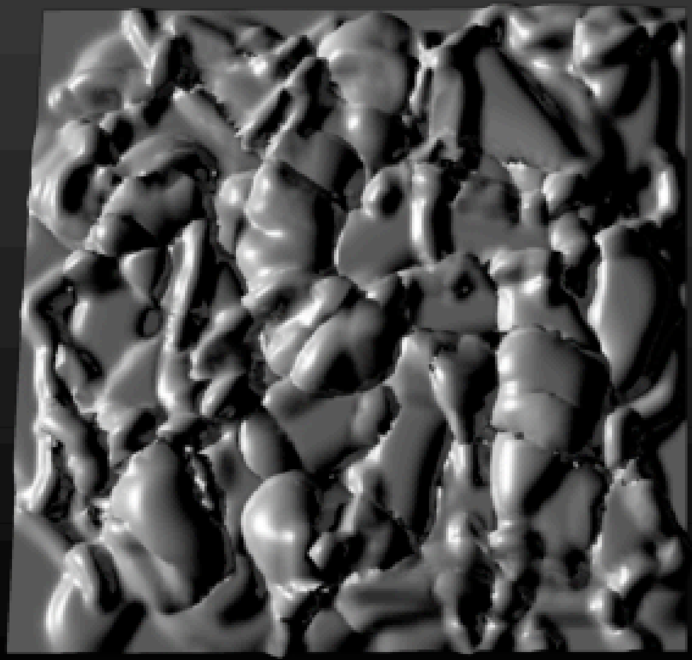


- Store the normal vector from the intersection points in a texture

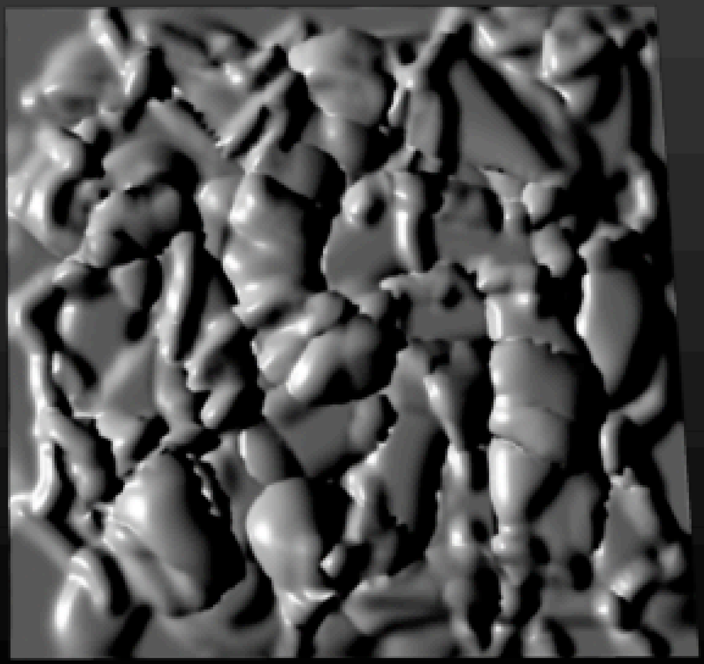


Normal Map Generation

- Render lo res surface + normal map



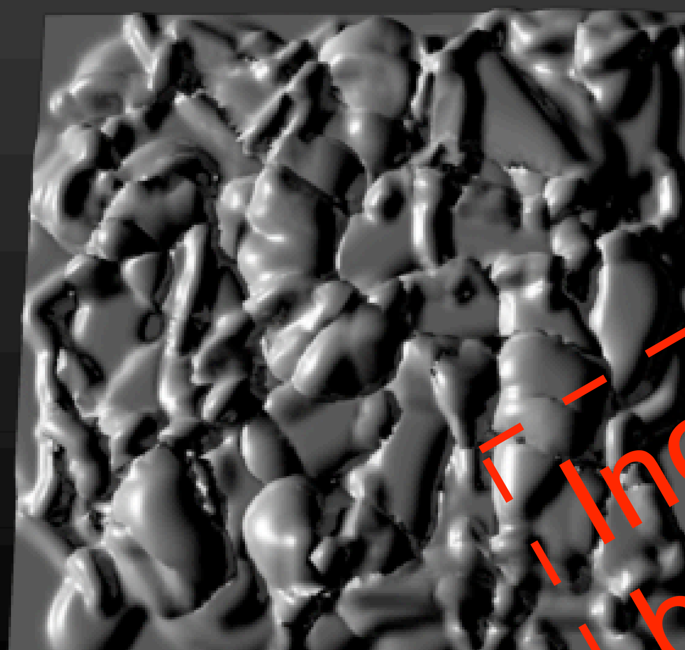
Hi res - 20k triangles



Lo res - 2 triangles
+ normal map

We need compression!

- Render lo res surface + normal map



Hi res - 20k triangles

Lo res - 2 triangles
+ normal map

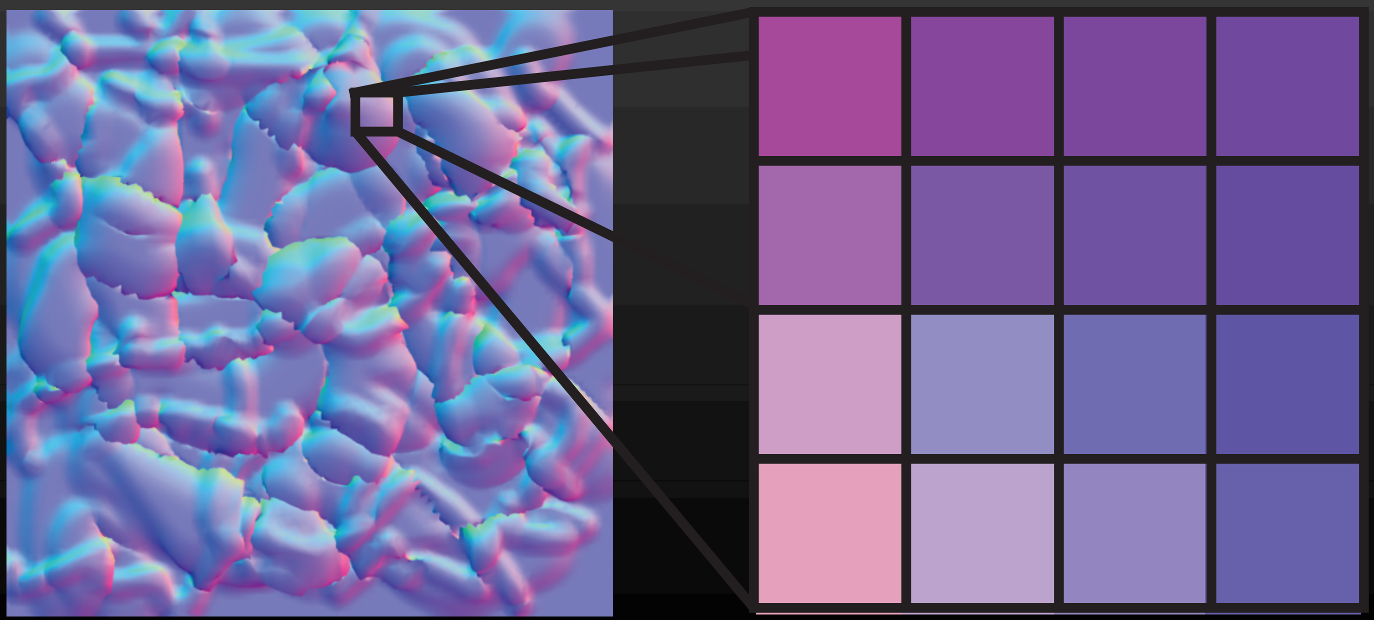
Increased Memory bandwidth!!!

Previous Work

- Surface normal compression
  - [Deering 1995] targeting geometry compression
  - Costly algorithm for HW ~ 12 bits per normal
- S3 Texture Compression / DXTC
  - Good for colors - not designed for normals
  - Visible artifacts (edges, subtle curvatures)
- 3Dc
  - Dedicated format for normals
  - 8 bits per texel

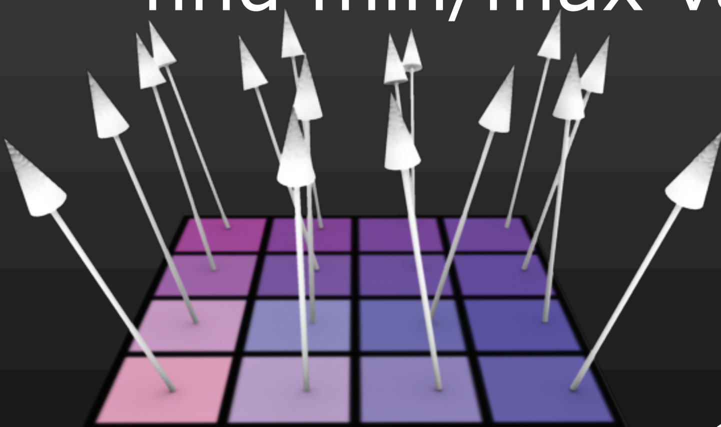
3Dc Overview

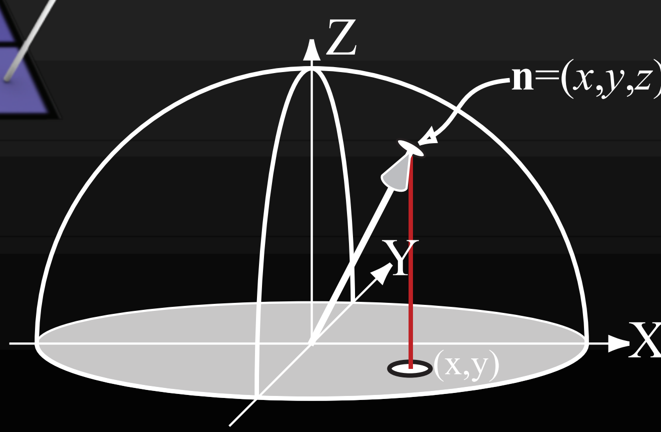
- Divide the input file in 4x4 blocks of texels

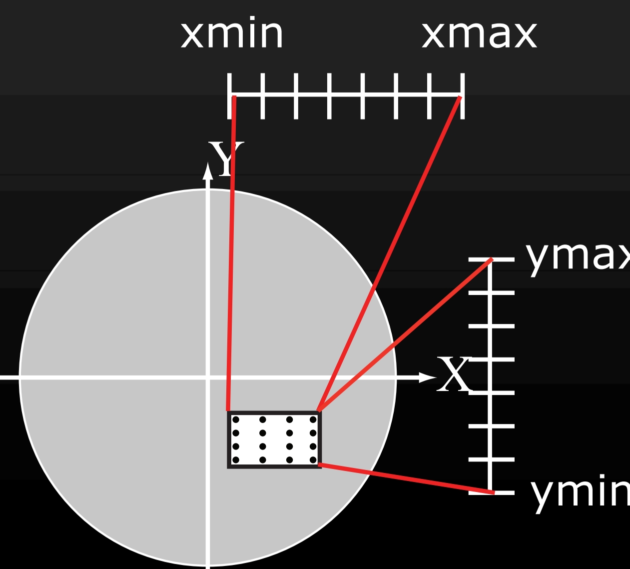


3Dc - Projection

- Project the normals on the xy plane and find min/max values of the bounding box

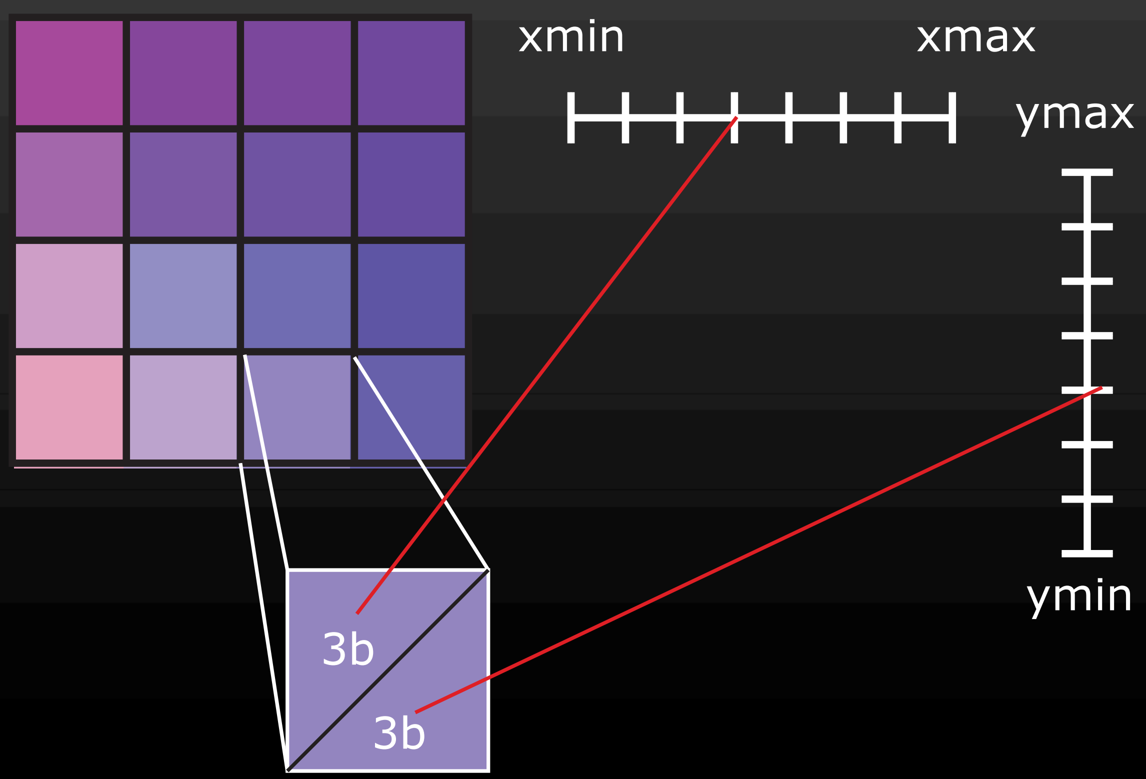






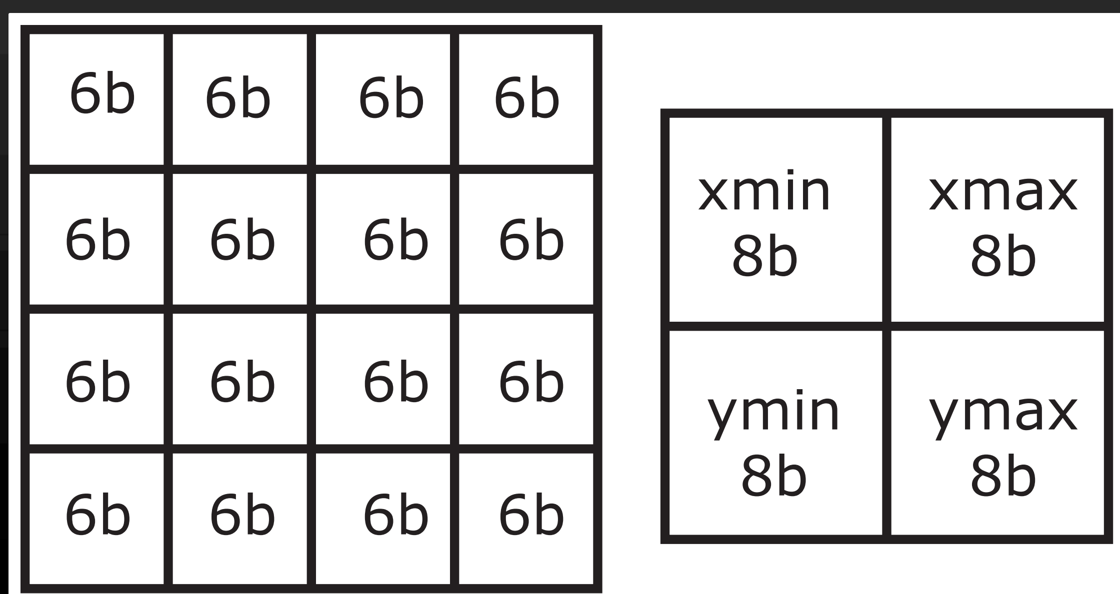
3Dc - Texel Quantization

- Map each texel to a quantized (x,y) value
  - Eight levels in x & y; (3,3) bits to select (x_i,y_i)



3Dc - Compressed Block

- Compressed form
  - 4x8 bits for x_{min}, x_{max}, y_{min}, y_{max}
  - 6x16 bits for per texel index
  - Total: 128 bits per block : 8 bits per texel



3Dc - Decompression

- Decompression
  - Reconstruct x & y from min/max values and texel indices.
  - Derive z from the unit length condition

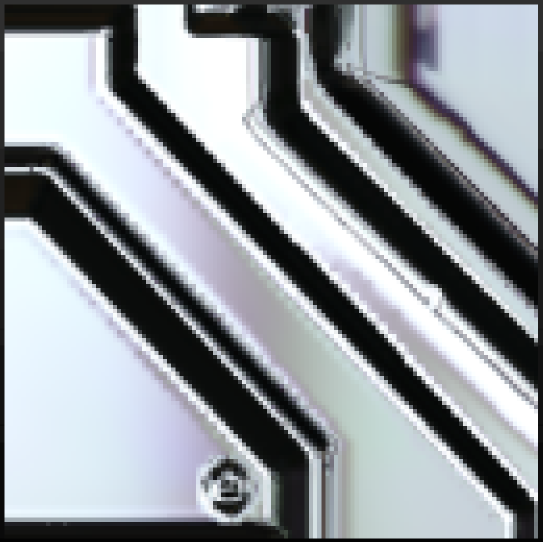
$$z = \sqrt{1 - x^2 - y^2}$$

  - Can be done in a pixel shader
- Supported by new [ATI] graphics cards

Problems with 3Dc

- Difficult scenarios
  - Slow gradients, sharp edges, directed features

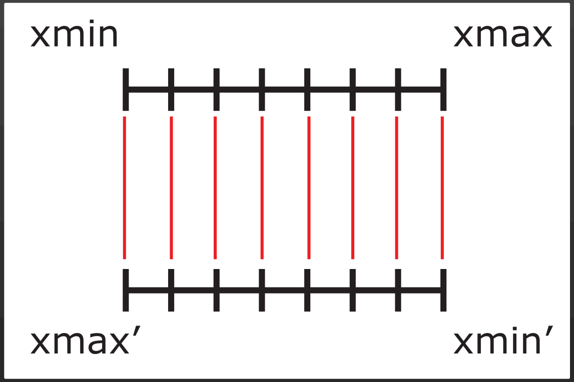
3Dc



Original

3Dc can be improved!

- Observation (used in DXT1)
  - Swap min & max values
    → same reconstruction levels
  - One bit unused per channel!
  - Use these to signal new modes!



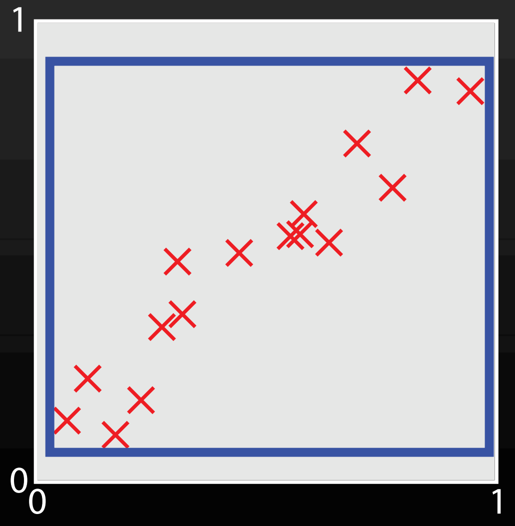
X	Y	mode
$x_{min} < x_{max}$	$y_{min} < y_{max}$	Standard 3Dc
$x_{min} \geq x_{max}$	$y_{min} < y_{max}$	?
$x_{min} < x_{max}$	$y_{min} \geq y_{max}$	?
$x_{min} \geq x_{max}$	$y_{min} \geq y_{max}$	?

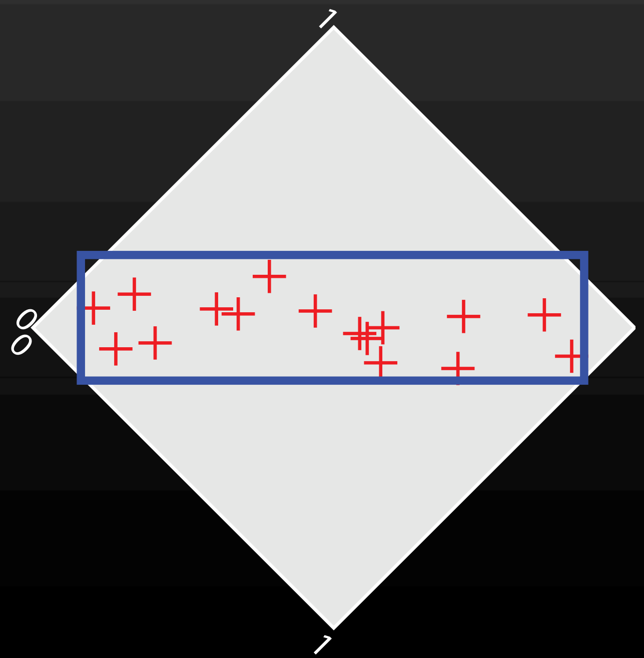
New techniques for 3Dc

- Rotation Compression
- Variable Point Distribution
- Differential Encoding

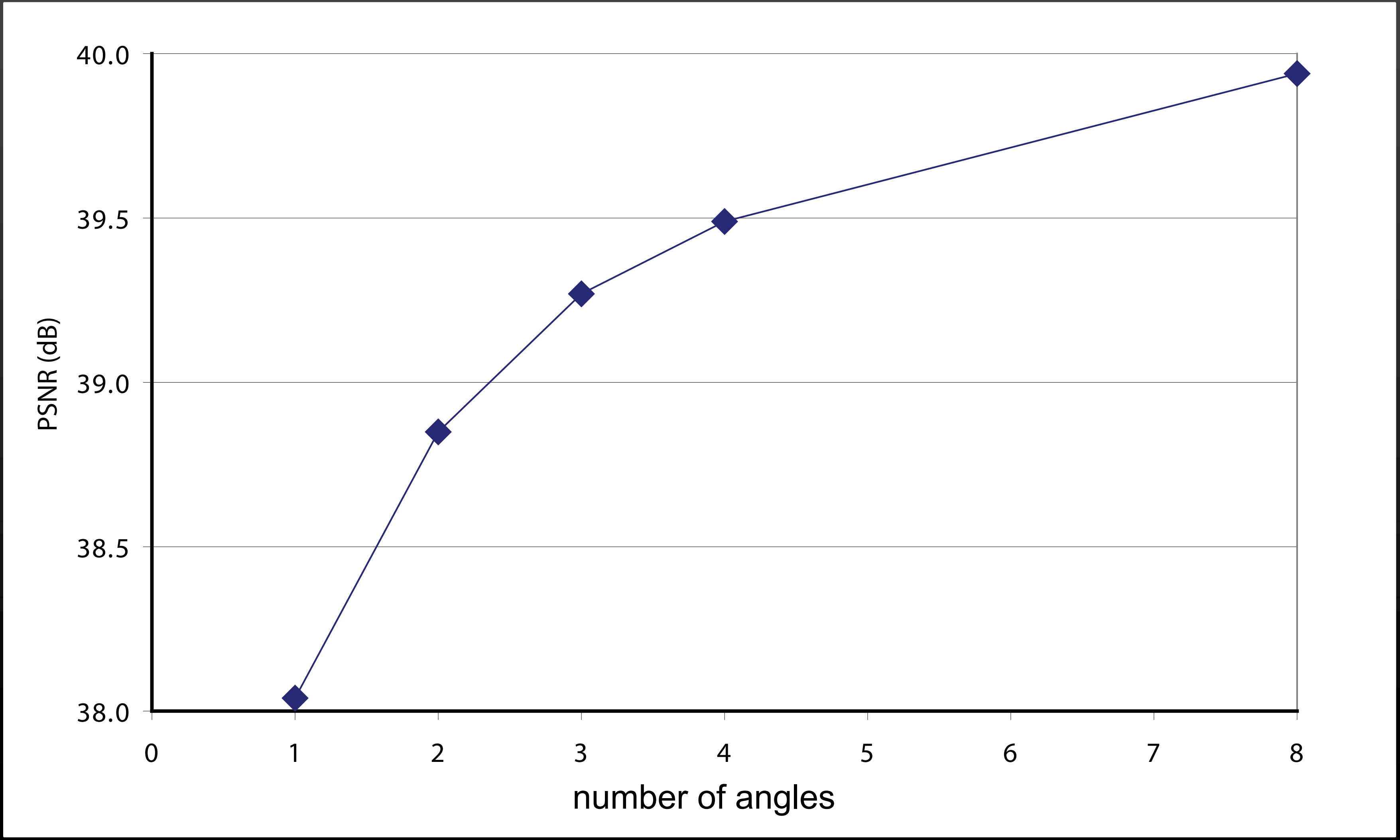
Rotation Compression

- Rotate coordinate frame for a more compact bounding box
- Storage cost: one angle per block



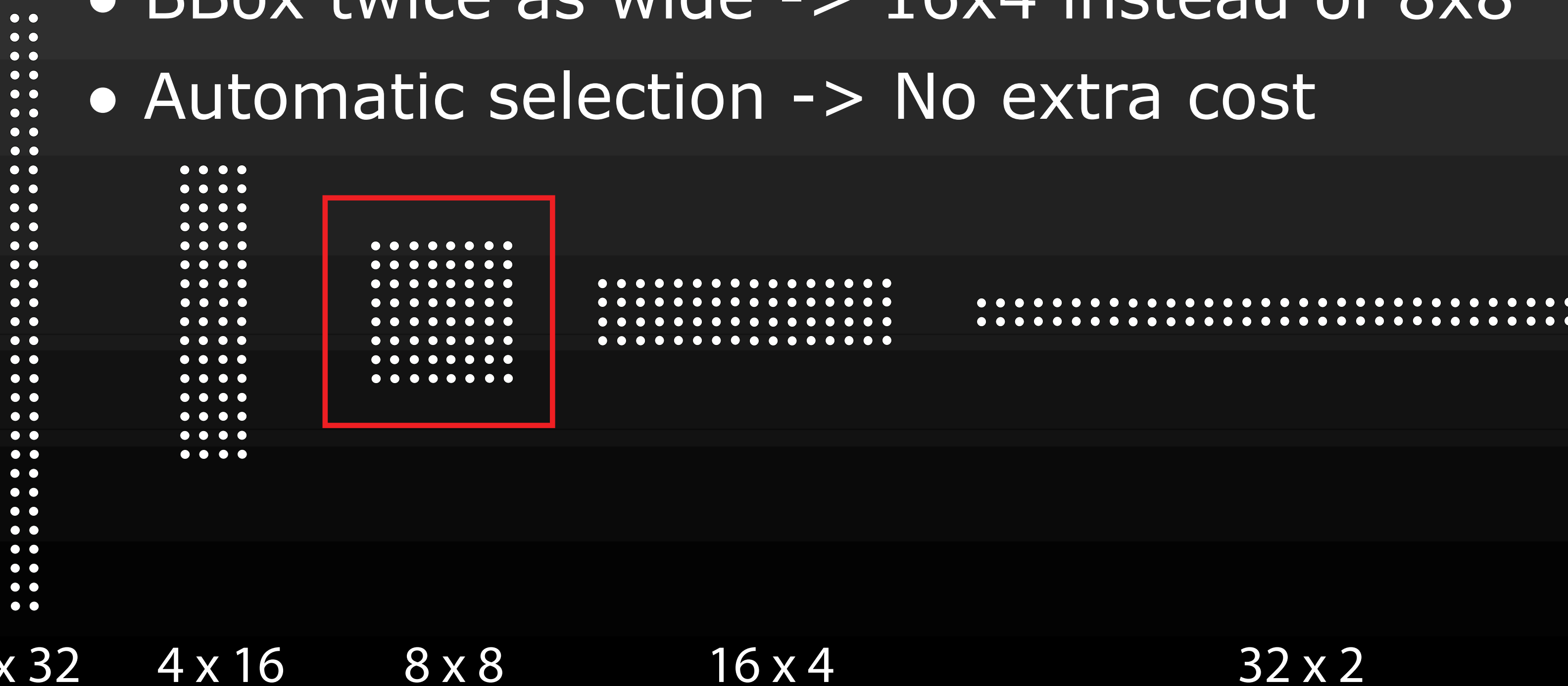


3Dc with Rotation

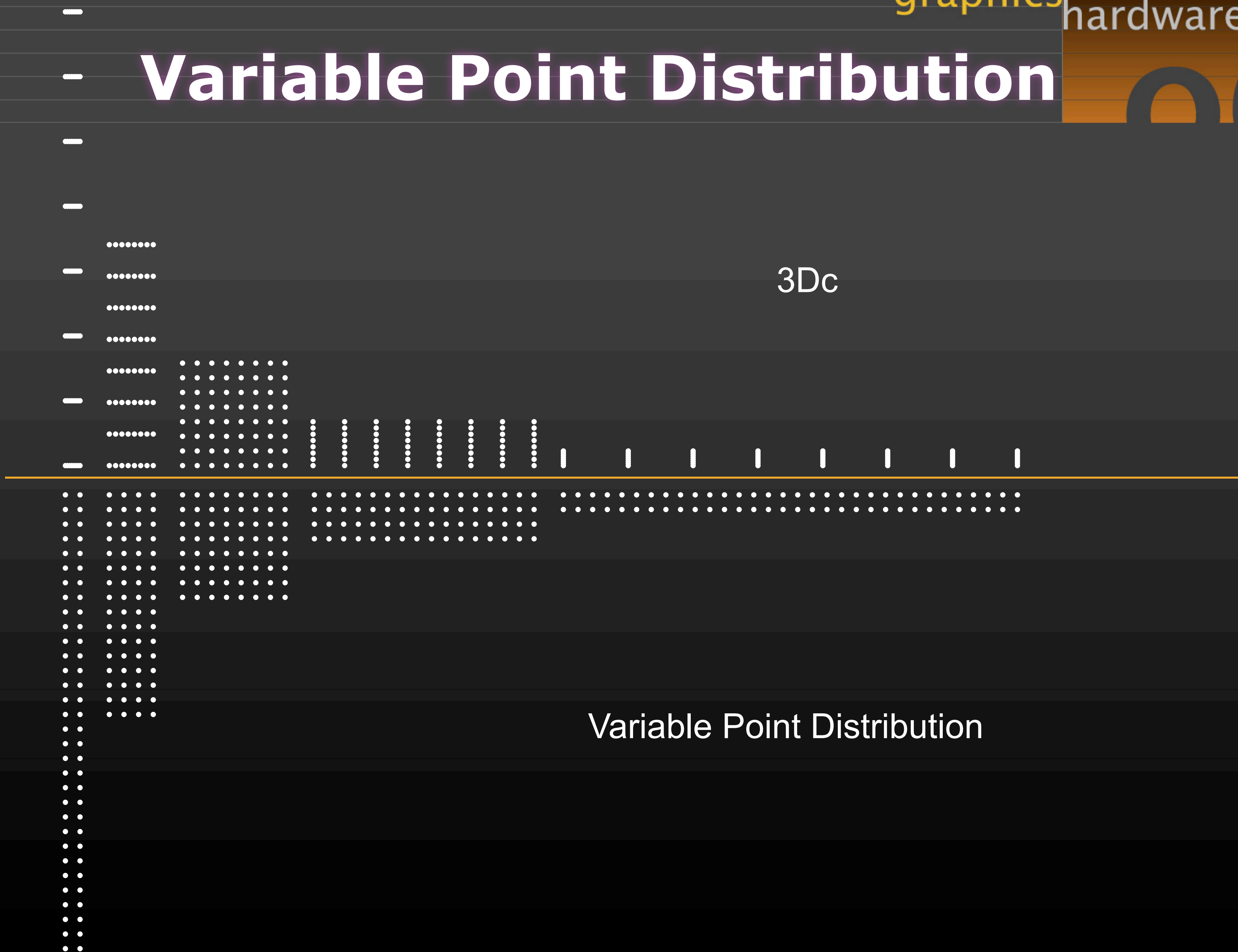


Variable Point Distribution

- 3Dc : points in a 8x8 grid
- Our approach : use aspect ratio of bbox
  - BBox twice as wide -> 16x4 instead of 8x8
  - Automatic selection -> No extra cost



Variable Point Distribution



Differential Encoding

- Slowly varying normals are problematic:
  - Smallest interval is too wide (range/255)

  - The interval cannot be placed accurately enough

- Reinterpret the bits differentially!
  - (xmin,xmax) → (x_{min}^*, Δ_x)

Differential Encoding

- Suppose we can reinterpret the bits!
- A suitable encoding for slow maps:
  - Use 11 bits (8.3) for x_{min}* and y_{min}* base values
  - 4 bits (2.2) for Δ_x and Δ_y
    - $x_{max} = x_{min}^* + \Delta x$, $y_{max} = y_{min}^* + \Delta y$
  - Smallest representable interval four times smaller
  - Location of an interval border encoded with three additional fractional bits

Mapping Function

- We want to reinterpret the bits of x_{start} & x_{stop} while preserving $x_{start} \geq x_{stop}$
- Numbering scheme:
  - Remap triangle $x_{start} \geq x_{stop}$ to upper rectangle
- Example
  - (2x3 bit values) 32 numbers in upper rectangle: 5 bits can be extracted
  - (2x8 bit) 15 bits extracted: 11 (base) + 4 (Δ)

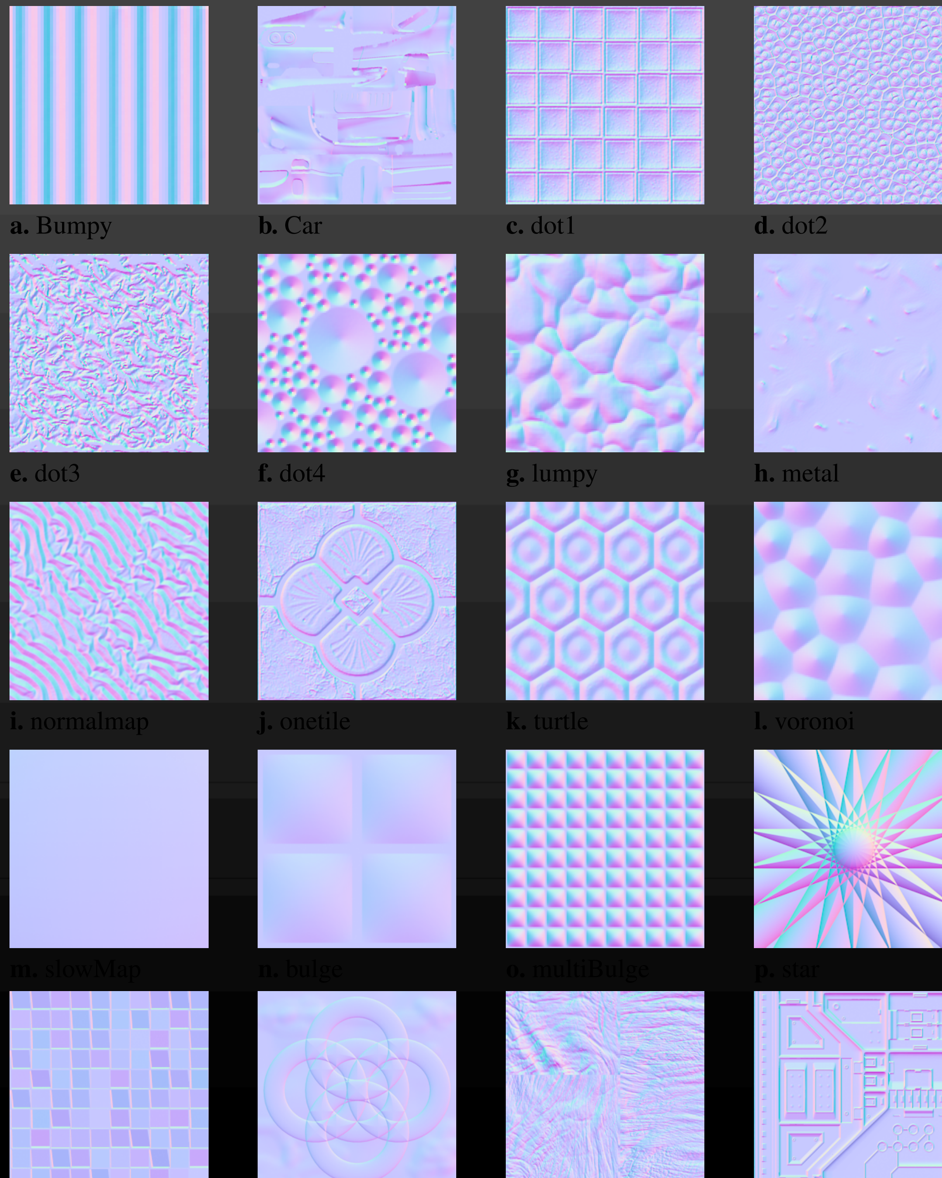
	x_{start} 0	1	2	3	4	5	6	7
x_{stop} 0	0	1	2	3	4	5	6	7
1	■	9	10	11	12	13	14	15
2	■	■	18	19	20	21	22	23
3	■	■	■	27	28	29	30	31
4	■	■	■	■	x	26	25	24
5	■	■	■	■	■	x	17	16
6	■	■	■	■	■	■	x	8
7	■	■	■	■	■	■	■	x

Combined Scheme

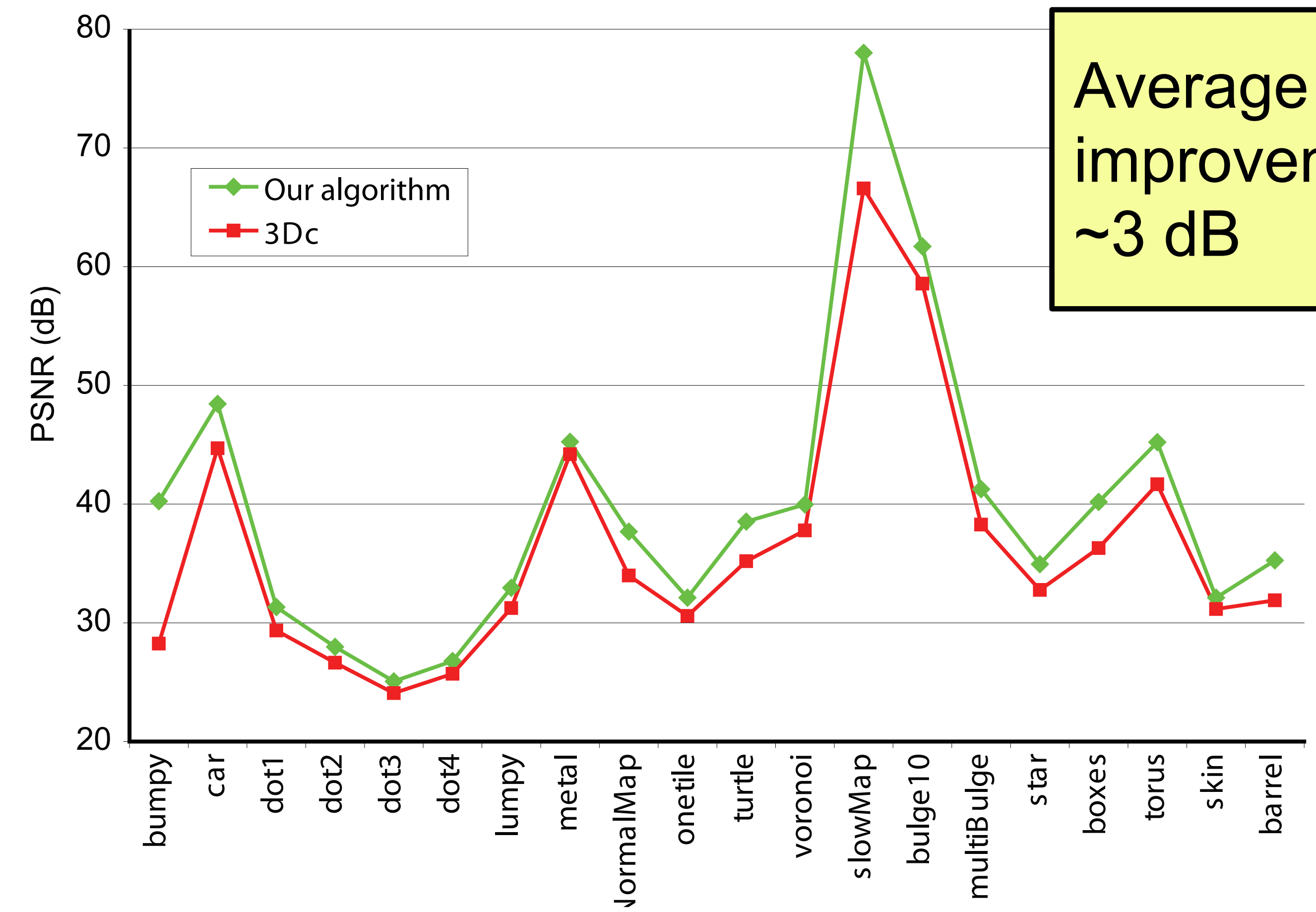
- Three rotations, variable point distribution and differential encoding
- Select modes by comparing x_{start}, x_{stop}, y_{start} & y_{stop}

mode	X	Y	bits	vpd
I: rot 0°	$x_{start} < x_{stop}$	$y_{start} < y_{stop}$	8+8	yes
II: rot 30°	$x_{start} \geq x_{stop}$	$y_{start} < y_{stop}$	8+8	yes
III: rot 60°	$x_{start} < x_{stop}$	$y_{start} \geq y_{stop}$	8+8	yes
IV: diff	$x_{start} \geq x_{stop}$	$y_{start} \geq y_{stop}$	8.3+2.2	no

Test Images


a. Bumpy
b. Car
c. dot1
d. dot2
e. dot3
f. dot4
g. lumpy
h. metal
i. normalmap
j. onetile
k. turtle
l. voronoi
m. slowMap
n. bulge
o. multiBulge
p. star

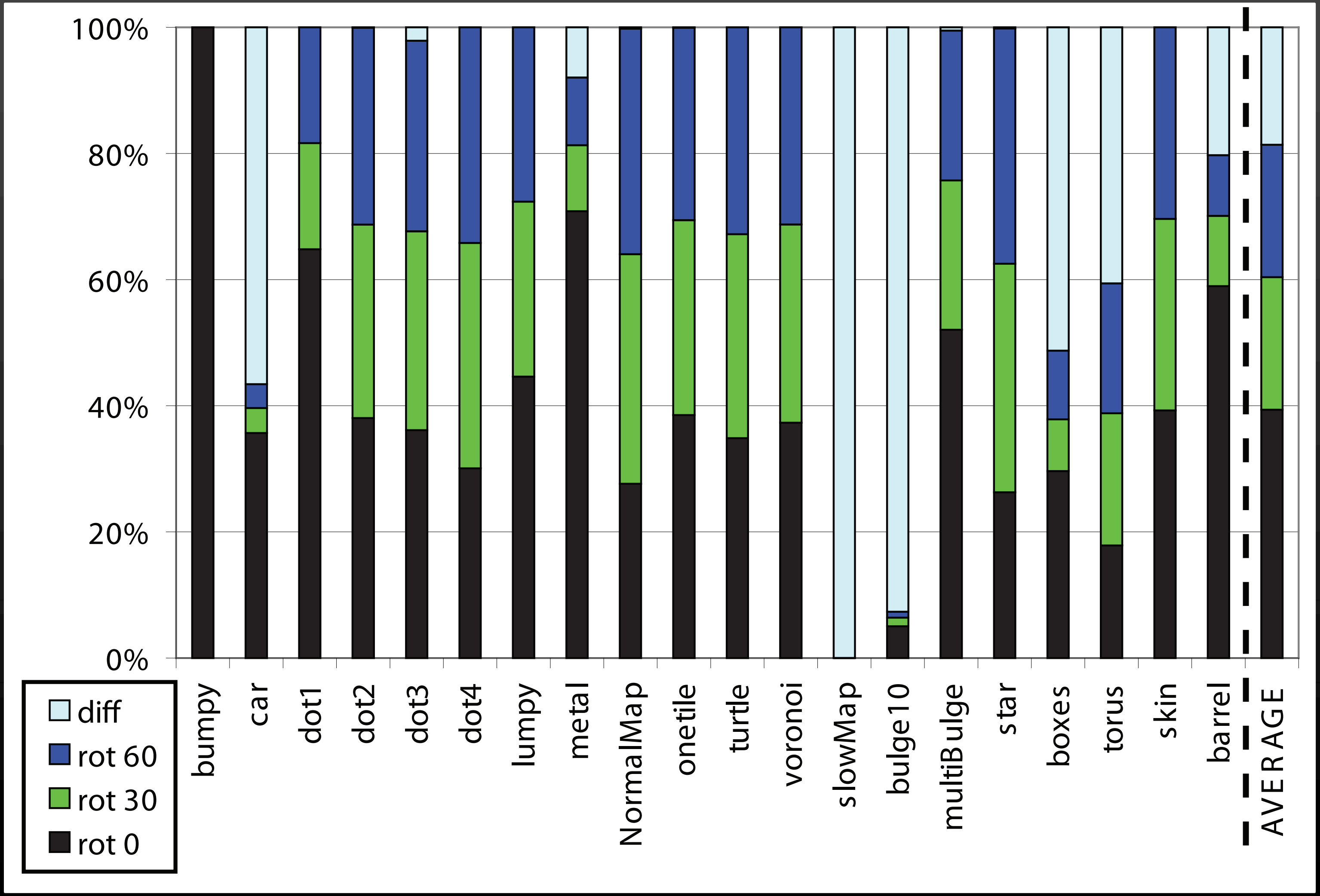
PSNR



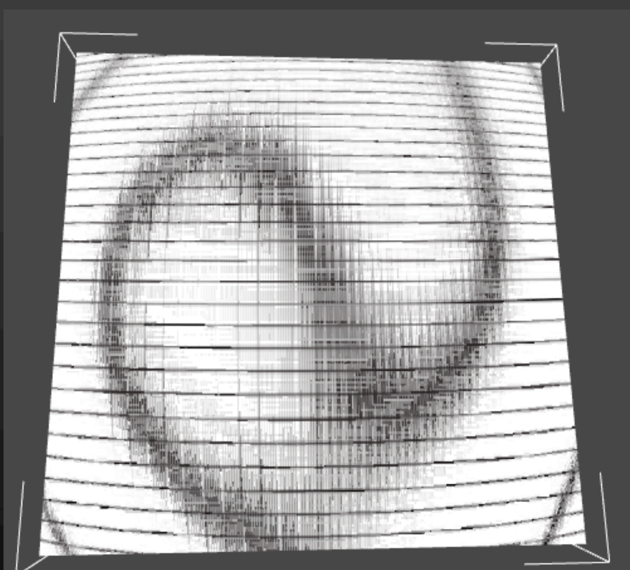
Average improvement: ~3 dB

graphics hardware 06

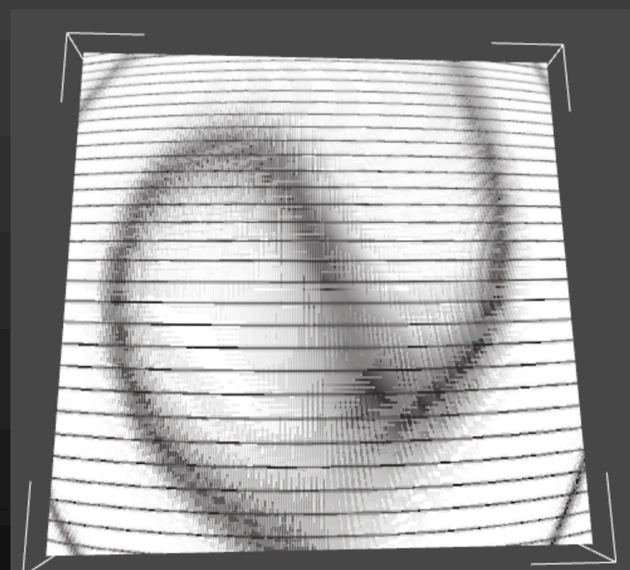
Frequencies



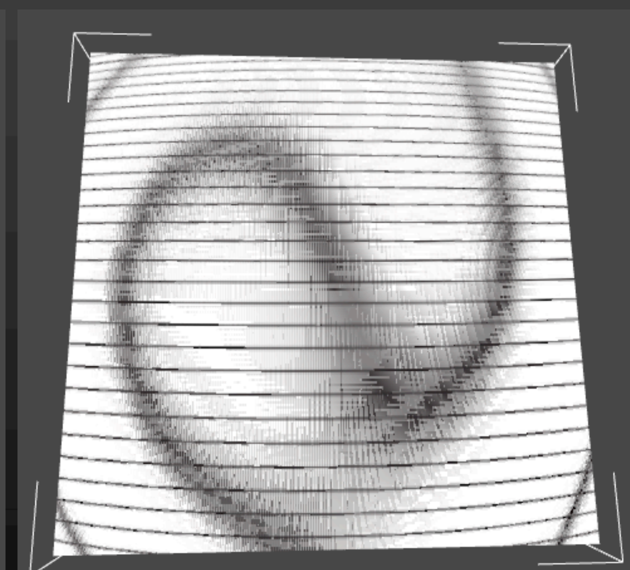
Slowly varying map example



3Dc

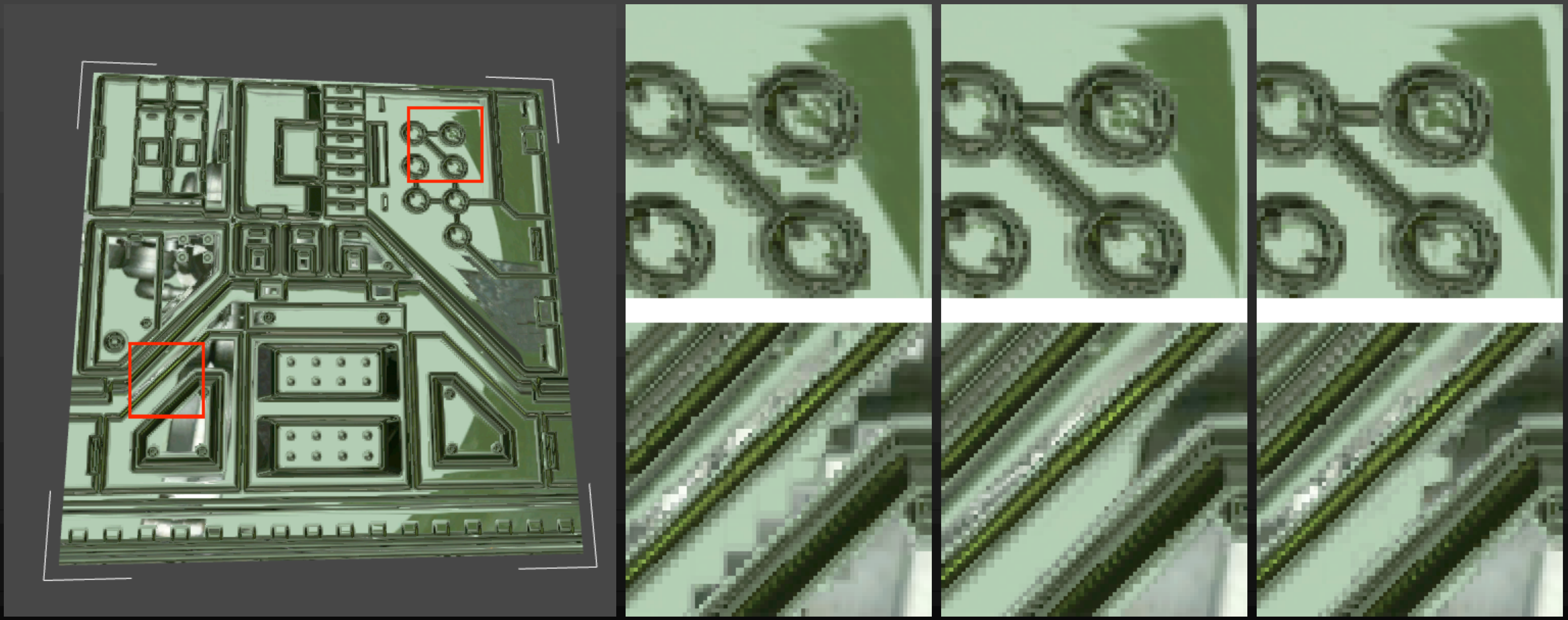


Original

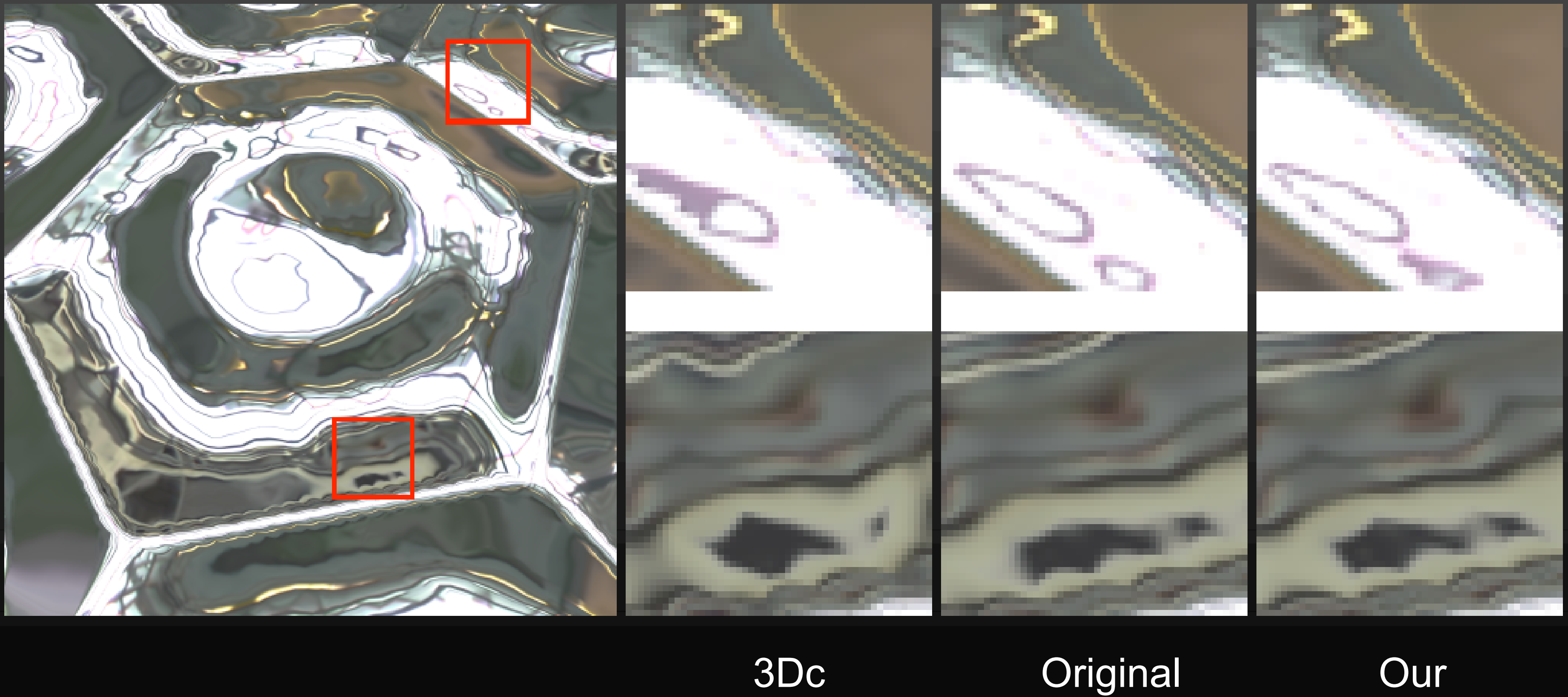


Our

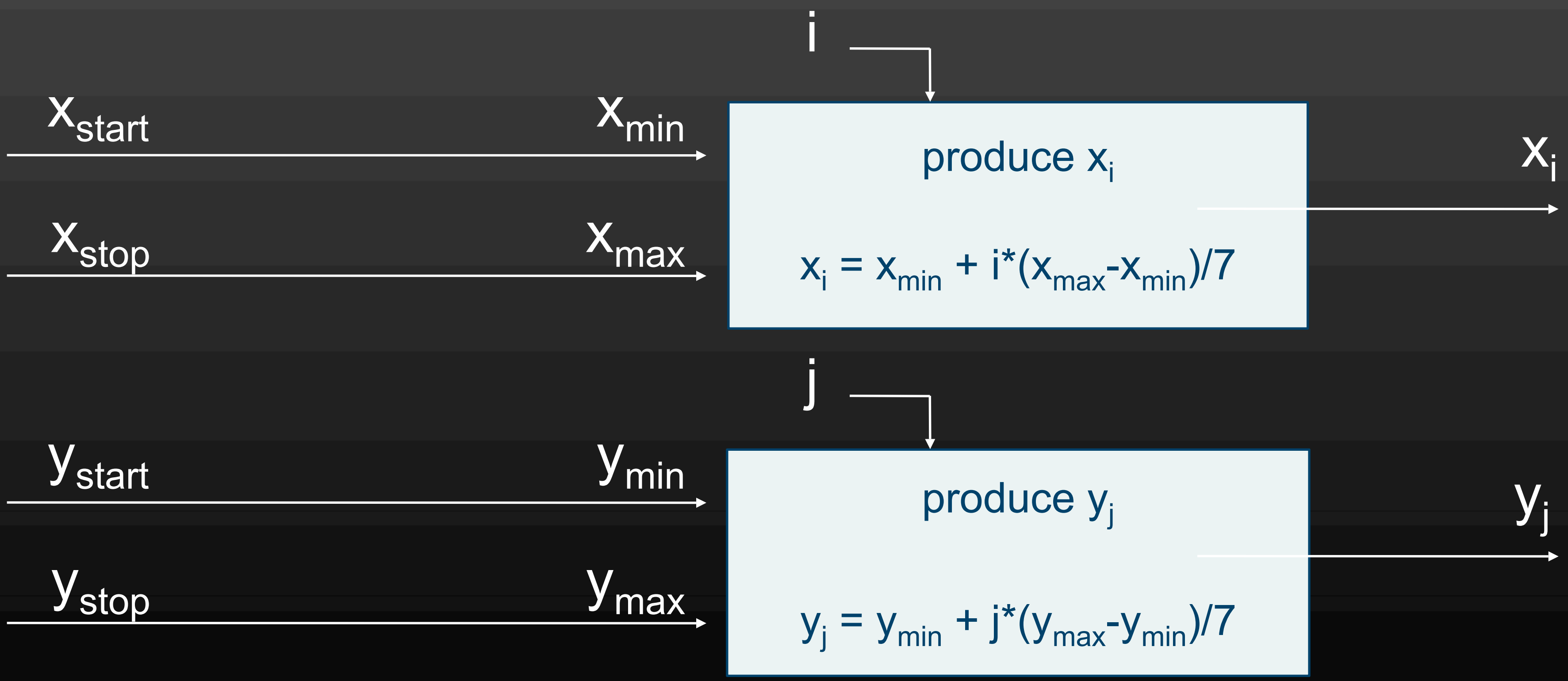
Result - Game texture



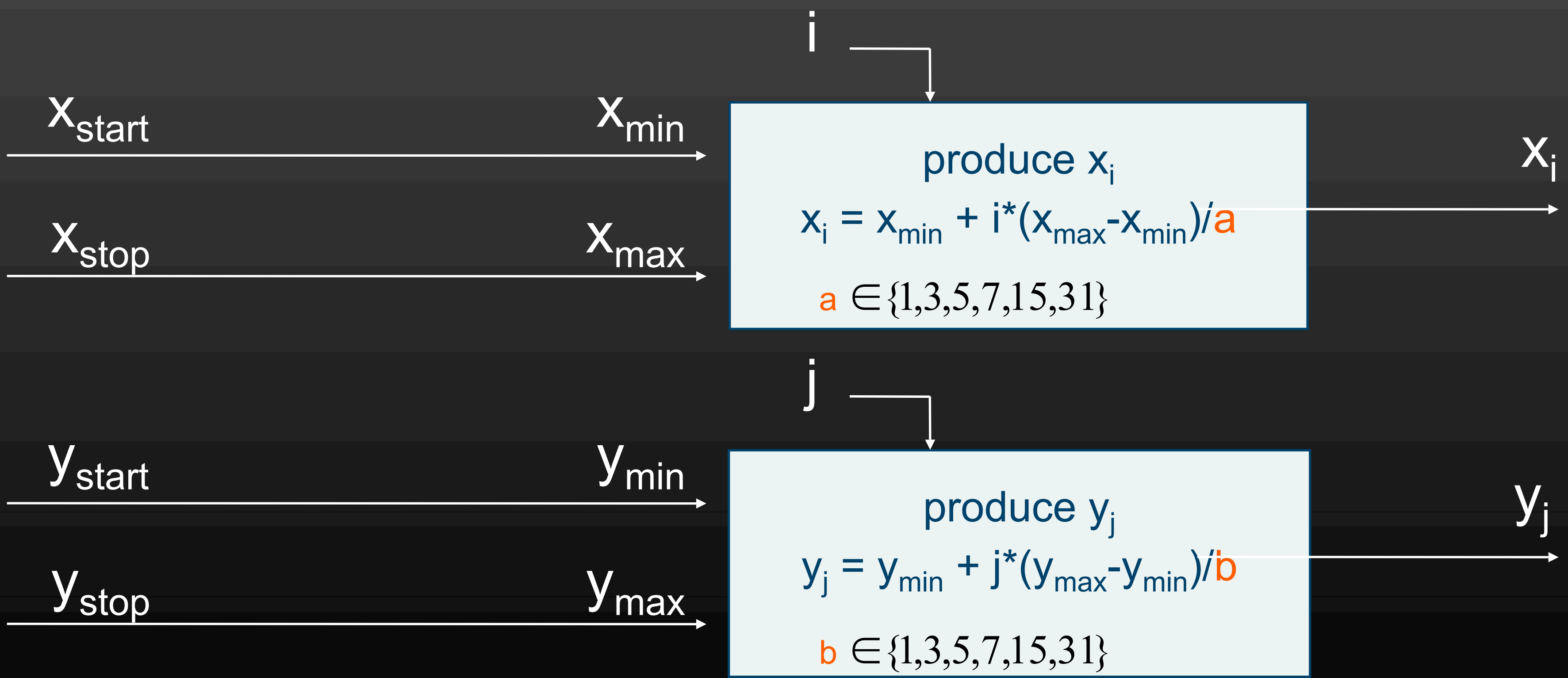
Results - Off-line rendering



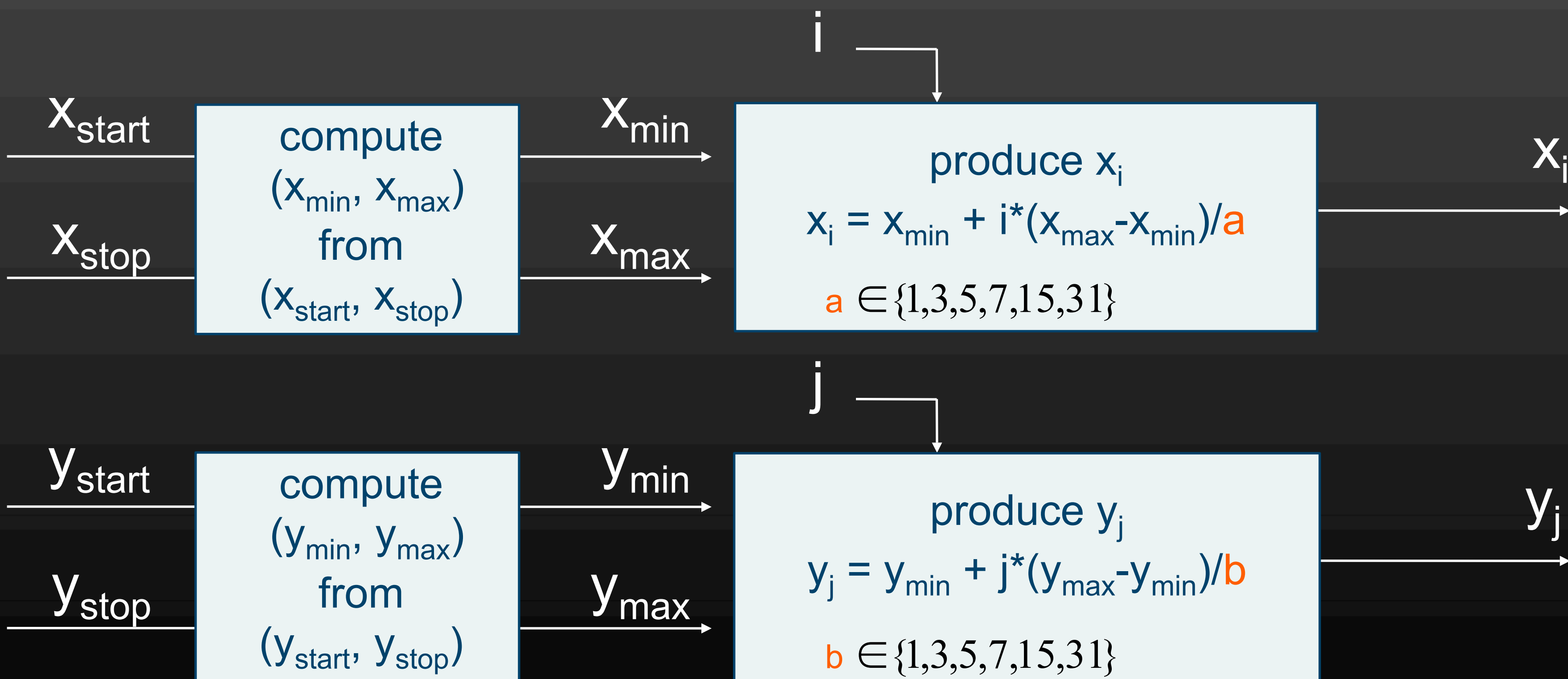
3Dc Decompression



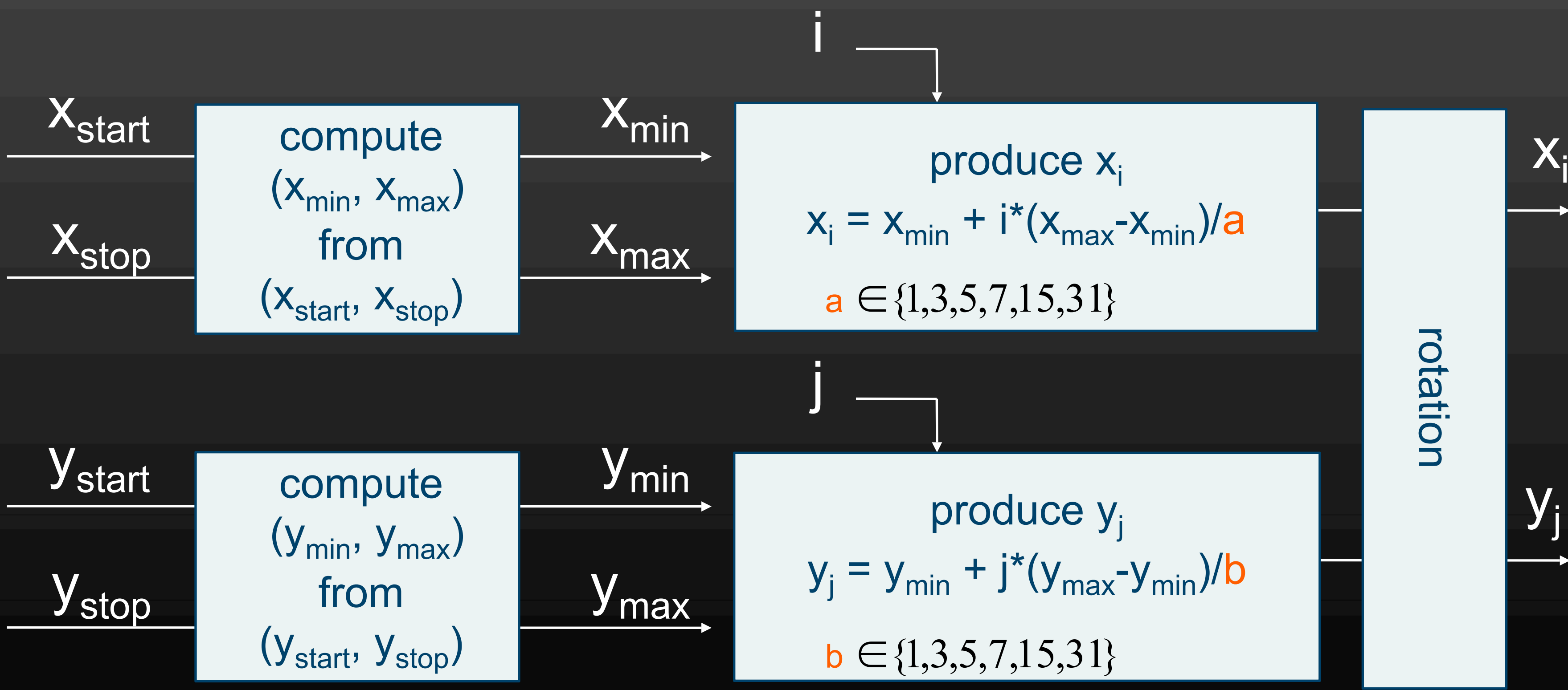
Variable Point Distribution



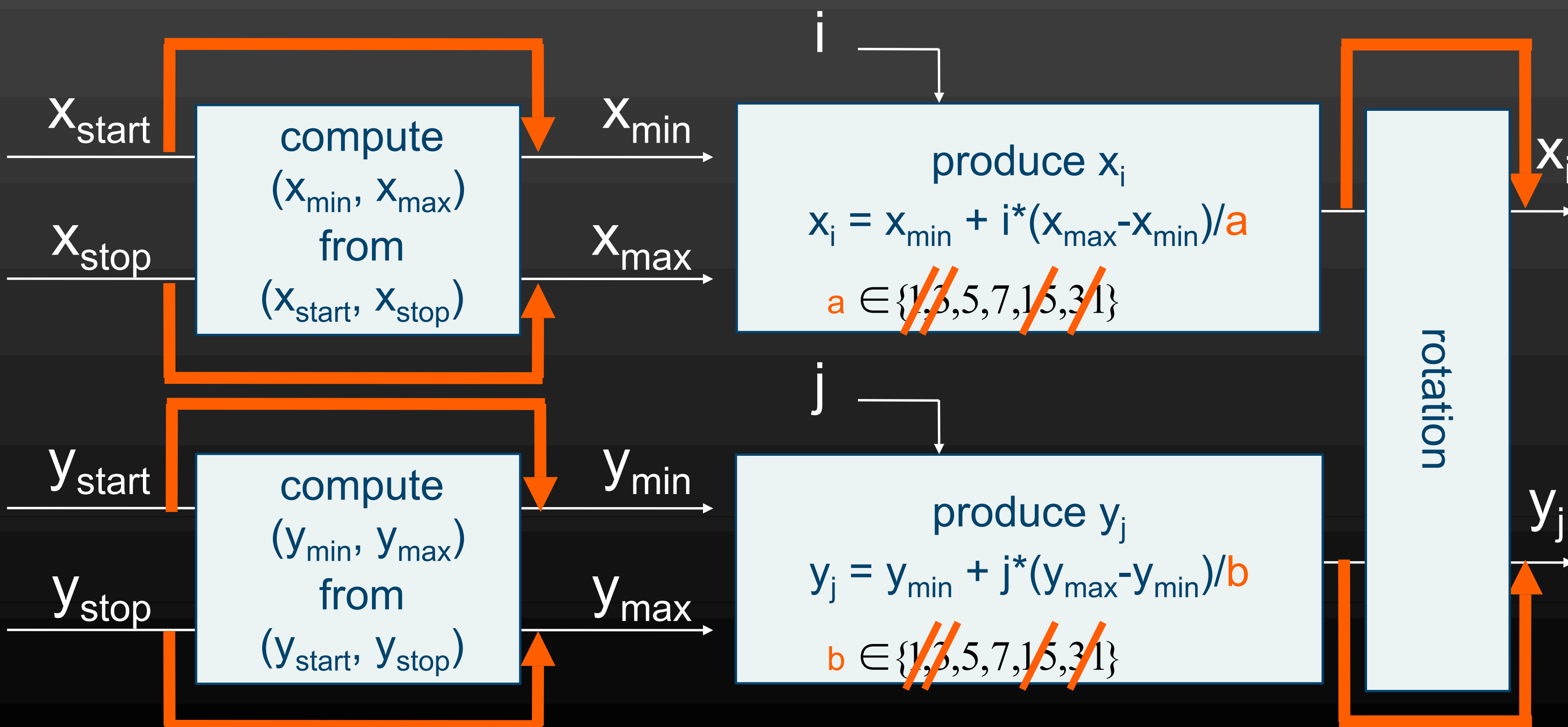
Differential Encoding



Rotation Decoding



Backward compatible with 3Dc and DXT5



Conclusions

- Higher quality than 3Dc
  - Still at 8 bits per texels
  - More flexibility with new modes
- Simple HW extensions
- Backwards compatible
  - 3Dc is a subset of our approach
  - DXT5 can be decoded with same HW
- API support?

Thank You!

- Swedish Foundation for Strategic Research (Mobile Graphics Grant)
- NVIDIA Fellowship
- ATI for making all the details of 3Dc openly available
- Pixologic
- Illuminate Labs
- Questions?

Average PSNR over all maps

mode	$\overline{PSNR}$ (dB)
3Dc	36.4
3Dc + Point Distr.	37.5
3Dc + Point Distr. + Rot	38.8
3Dc + Point Distr. + Rot + Diff	39.4

Mapping Examples (5 bit)

- Upper half of rect:

  $x_{start} = 6$, $x_{stop} = 2$

  - $v = (x_{stop} << 3)$ OR x_{start}

    $v = 22$

- Lower half of rect:

  $x_{start} = 7$, $x_{stop} = 5$

  - $v = (\text{NOT}(x_{stop}) << 3)$ OR $\text{NOT}(x_{start})$

    $v = 16$ (NOT = bitwise inversion)

- Encoding: x_{start} lower part of v, x_{stop} upper part. Invert both if $x_{stop} > x_{start}$

		x_{start}							
		0	1	2	3	4	5	6	7
x_{stop}	0	0	1	2	3	4	5	6	7
	1	■	9	10	11	12	13	14	15
	2	■	■	18	19	20	21	22	23
	3	■	■	■	27	28	29	30	31
	4	■	■	■	■	x	26	25	24
	5	■	■	■	■	■	x	17	16
	6	■	■	■	■	■	■	x	8
	7	■	■	■	■	■	■	■	x

Variable Point Distribution

aspect ratio ($a = \frac{y_{max} - y_{min}}{x_{max} - x_{min}}$)	distribution ($d_x \times d_y$)
$a < 1/8$	32×2
$1/8 \leq a < 1/2$	16×4
$1/2 \leq a \leq 2$	8×8
$2 < a \leq 8$	4×16
$a > 8$	2×32