graphics hardware 06

Efficient Depth Buffer Compression

Jon Hasselgren
Tomas Akenine-Möller

Lund University

Introduction

- Survey of efficient depth buffering
- A new depth compression algorithm

Why depth buffering?

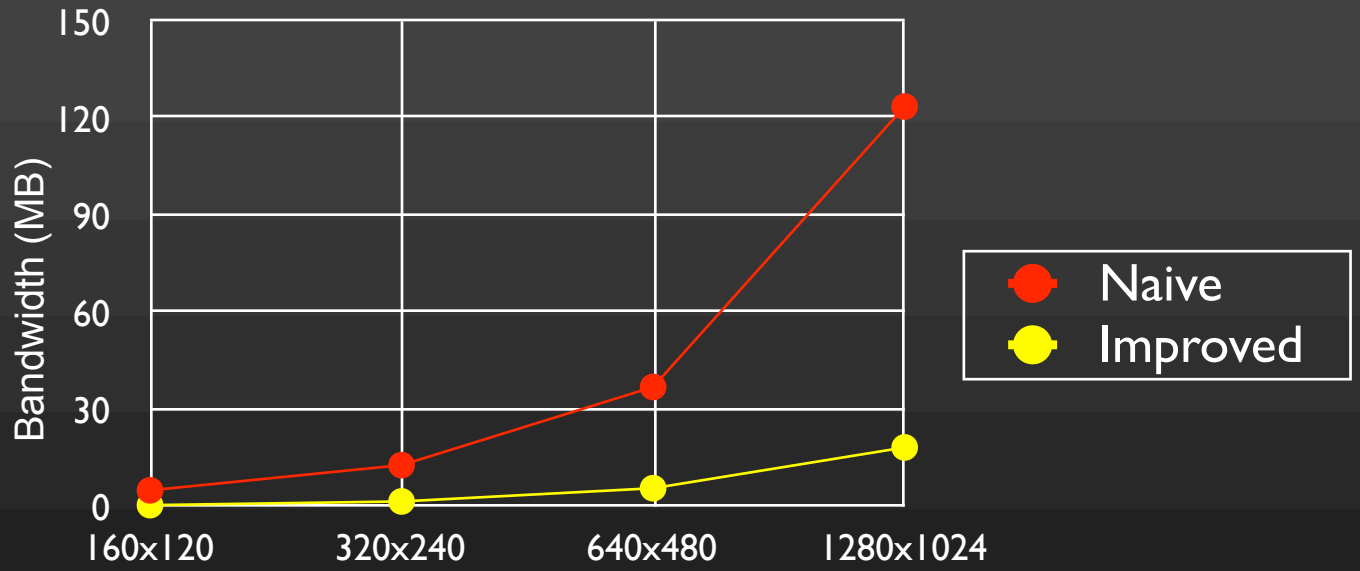
- ”... the brute-force approach which is already ridiculously expensive” [Sutherland et. al 77]

- But
  - Memory is ”free” nowadays
  - Simple algorithm
  - Easy to parallelize – Perfect for hardware

Brute Force

- It is still a brute force algorithm!
  - Naive depth buffering
- Improvements
  - Tiling
  - Caching
  - Hierarchical z culling
  - Depth compression

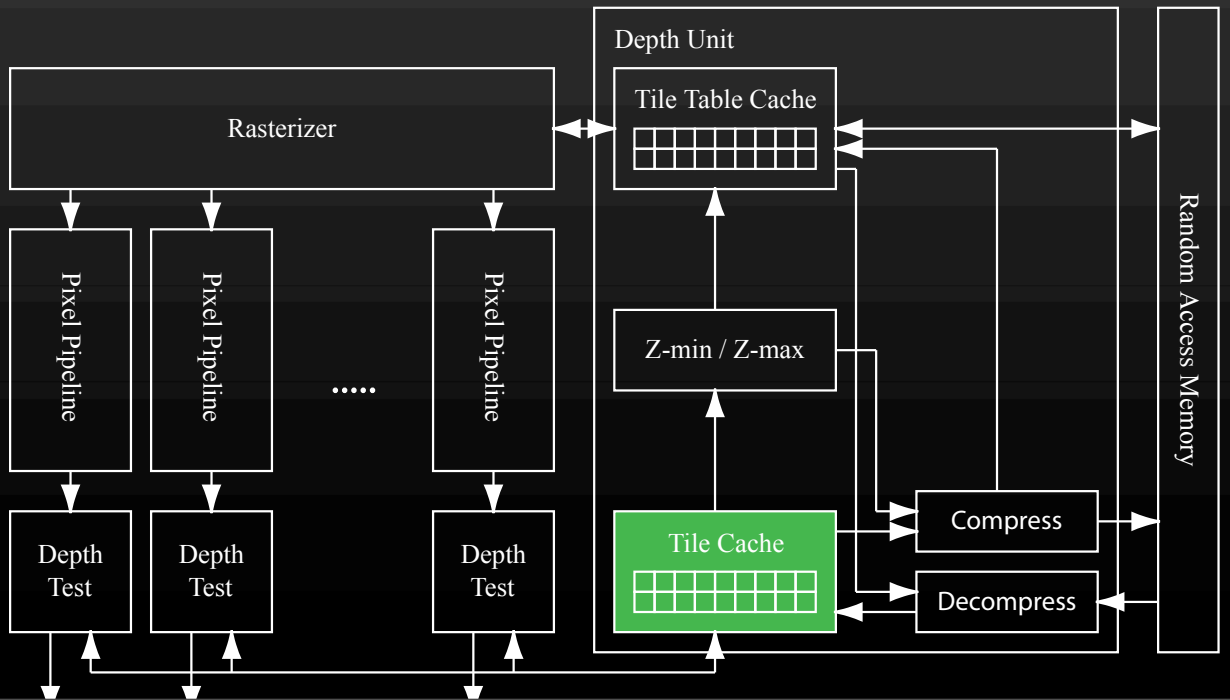
Naive vs Improved



- Up to 10x less bandwidth consumption
- Memory bandwidth is (almost) always a bottleneck

Tiled Depth Buffer

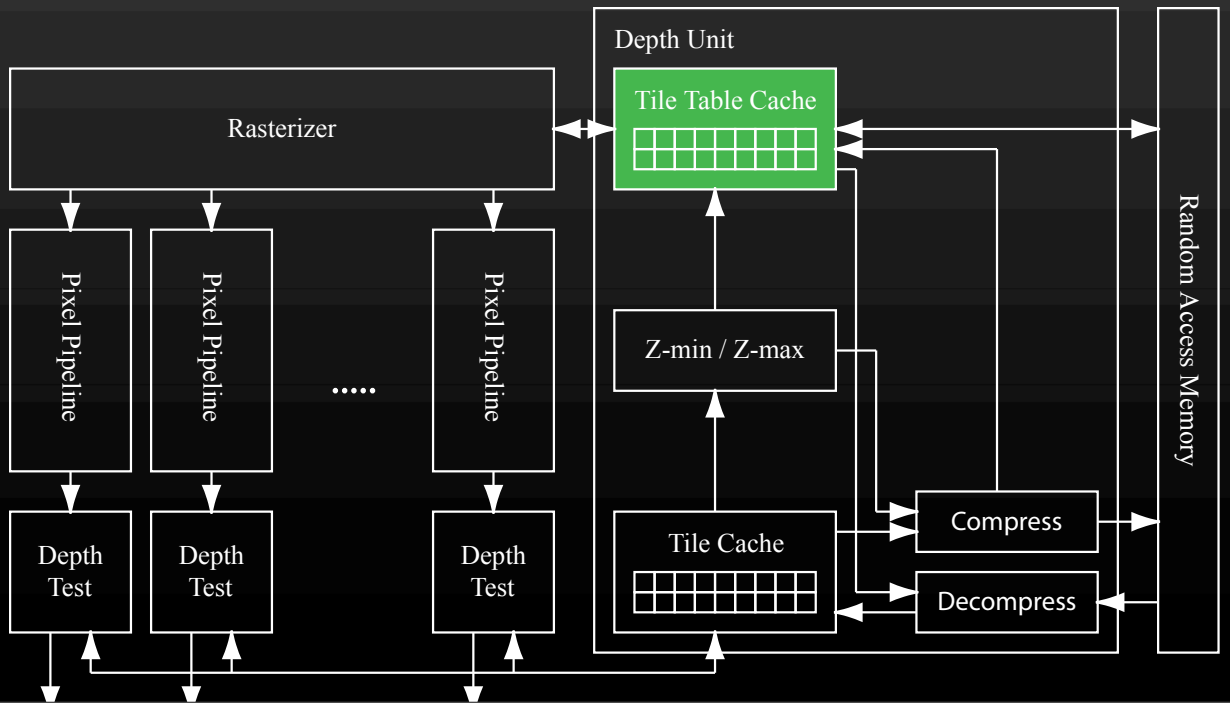
- Divide depth buffer into tiles
  - Small block of pixels
- A small cache memory with most recent tiles
  - Efficiency grows with smaller triangles



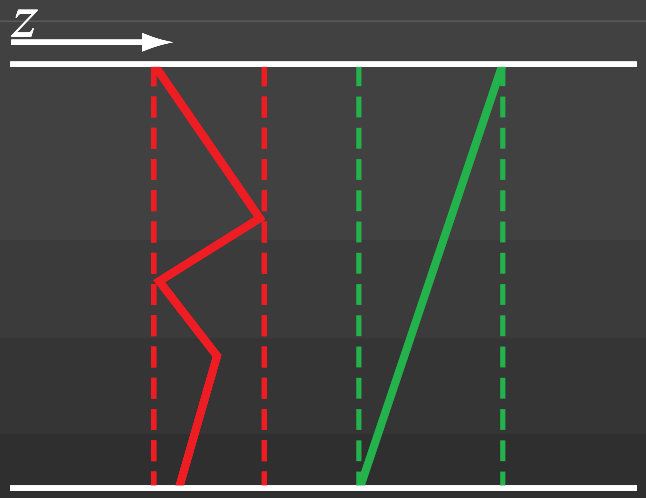
Tile Table [Morein 03]

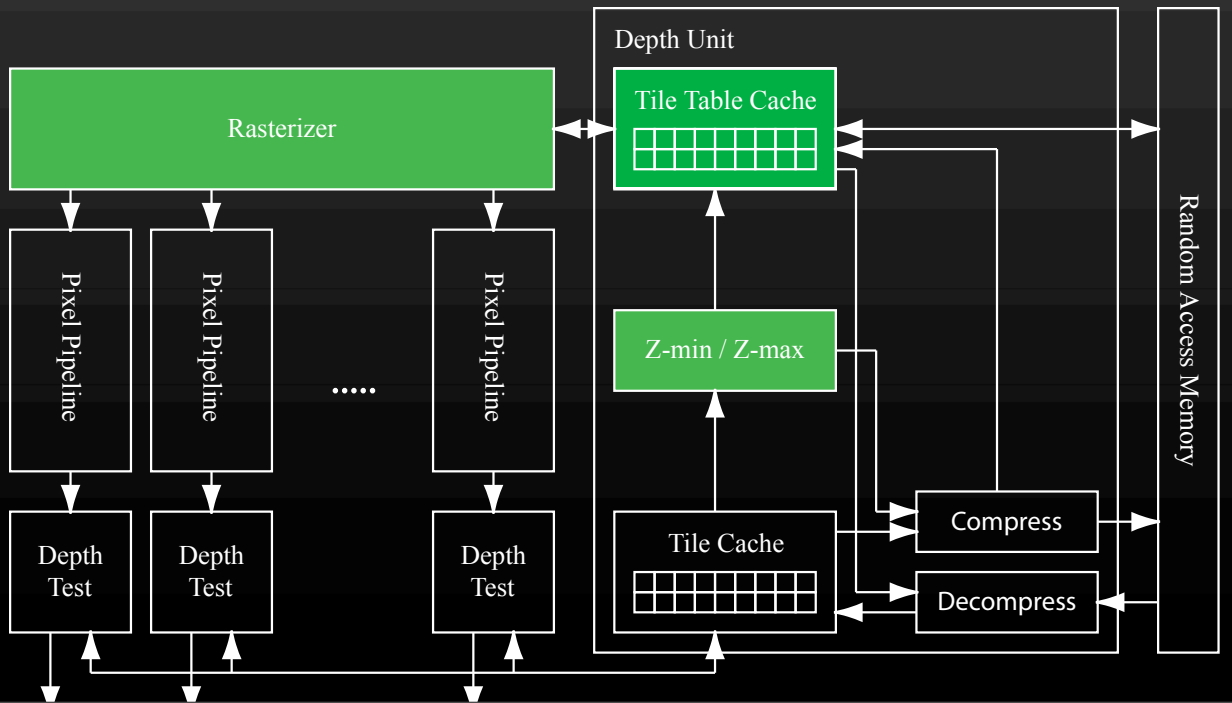
- One entry per tile
- ”Header” information
- Accessed through cache

- Examples:
  - Compression mode
  - Min/Max z value of the tile



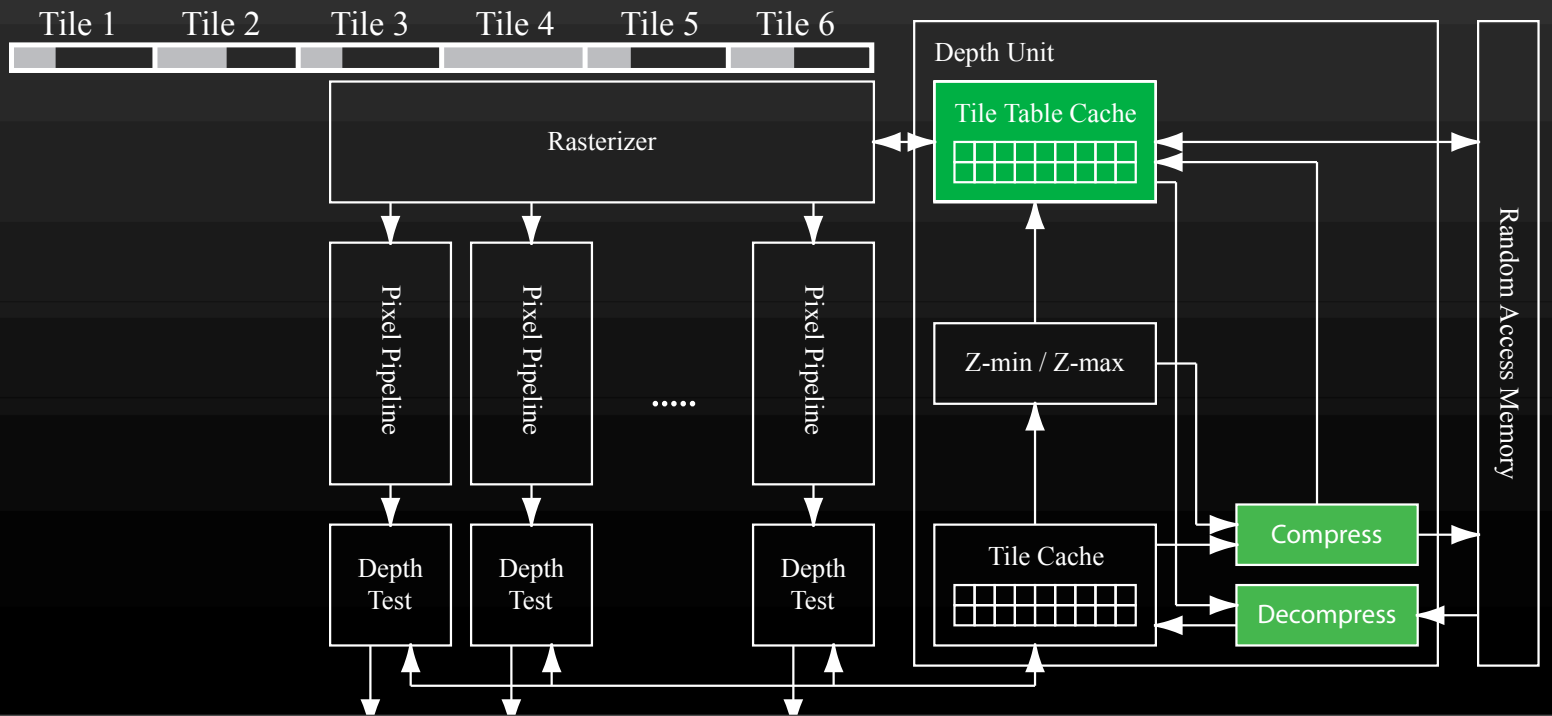
Hierarchical Culling [Morein00, Akenine-Möller03]





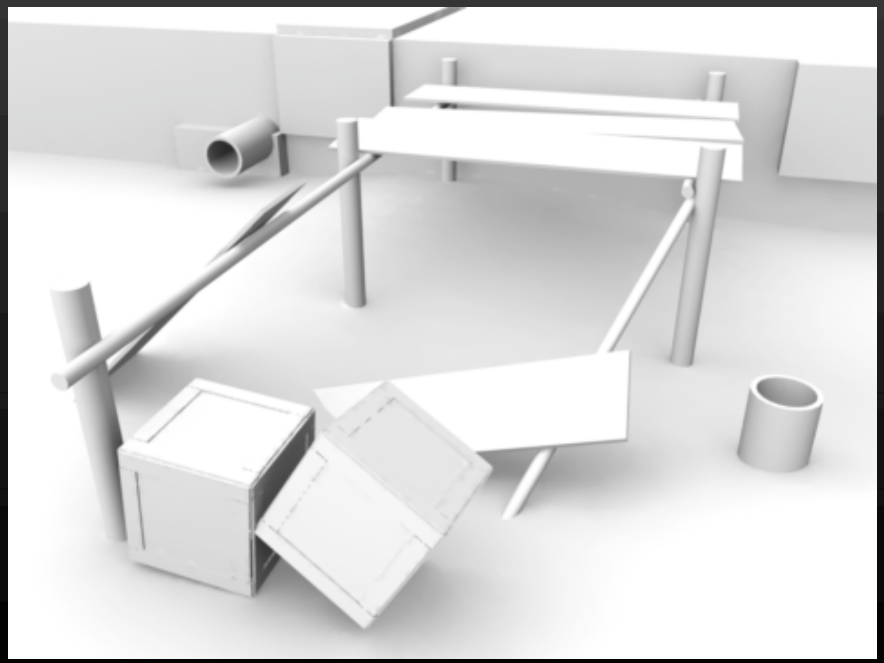
Compression

- Lossless compression!
  - Uncompressed fallback
  - Allocate memory for uncompressed data
- Done on a tile basis
- Fast (de)compression

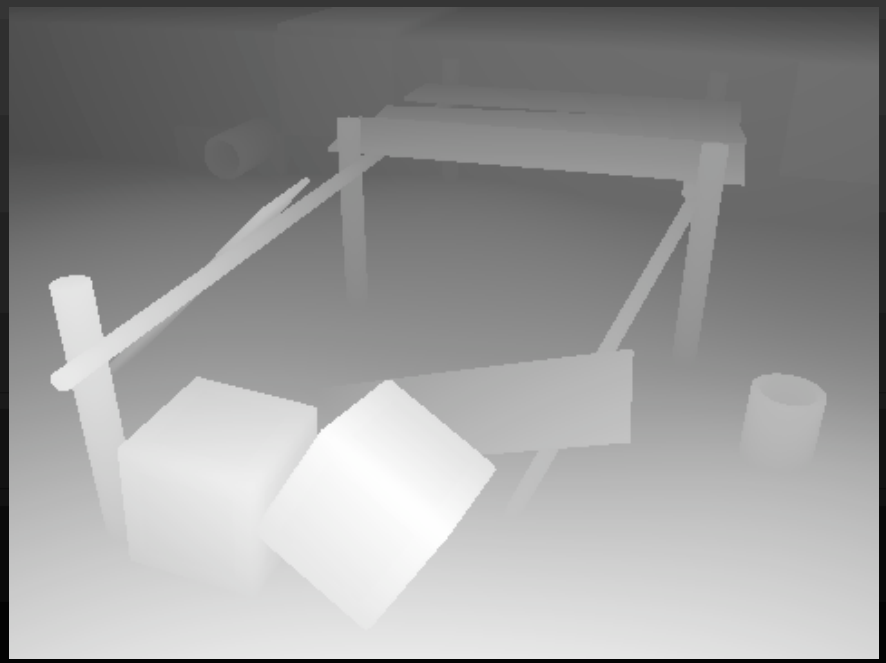


Depth Compression Algorithms

- Depth values are quite easy to compress
  - Smooth transitions and discrete edges

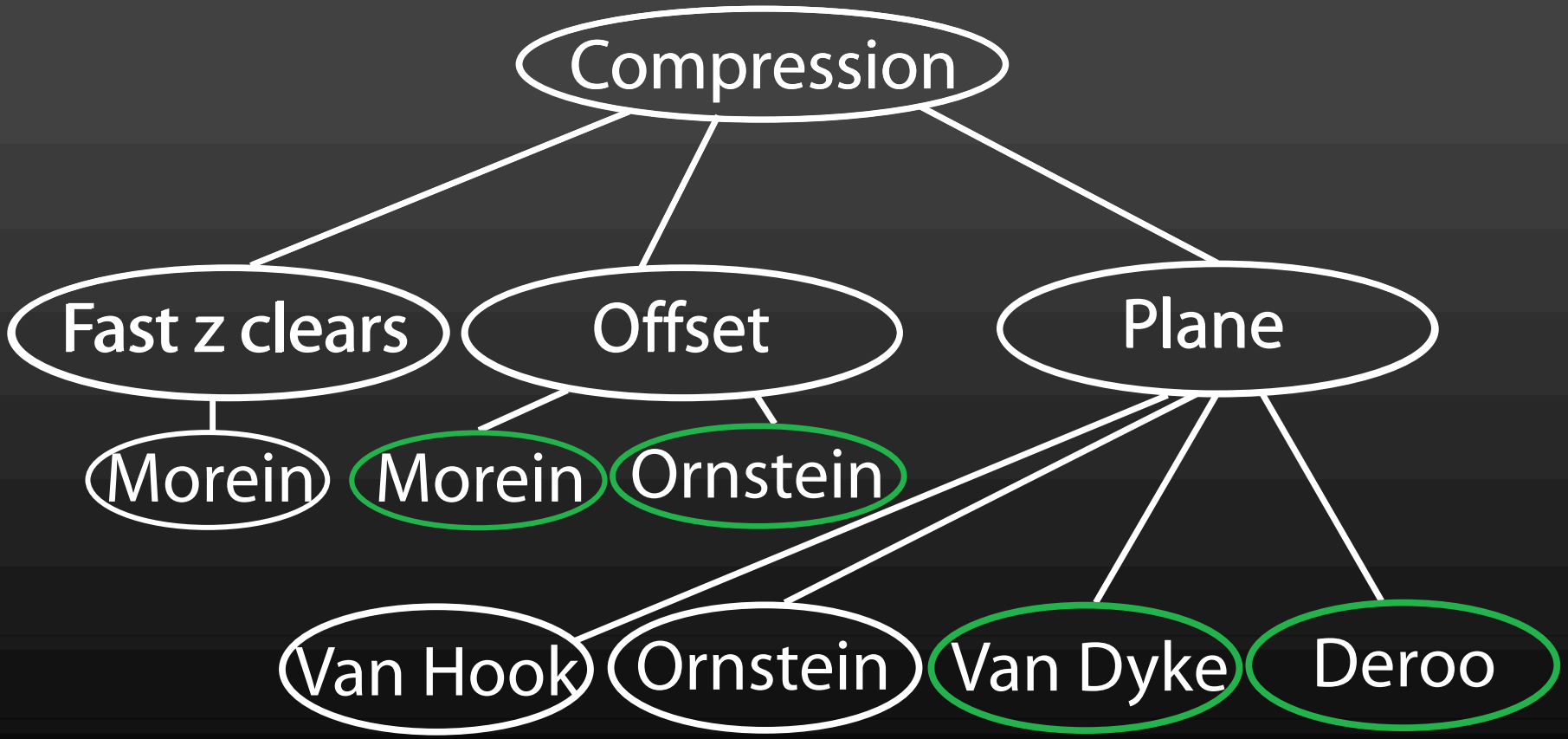


Rendered scene



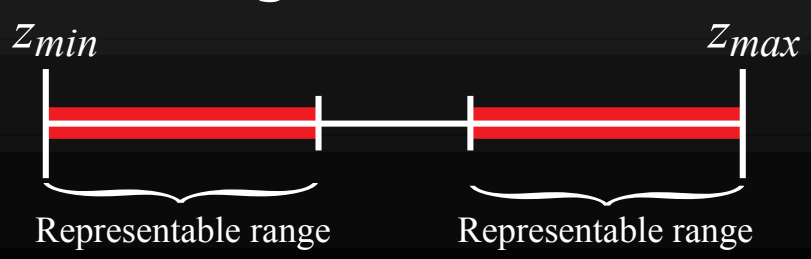
Depth buffer

Depth Compression Algorithms

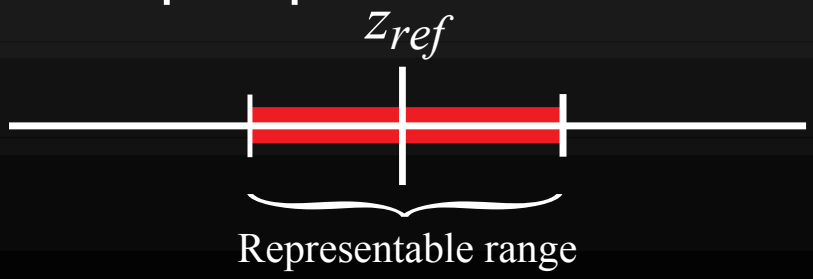


Offset Compression

- Select reference depth values
  - One or more per tile
  - Min / Max / Some predetermined pixel
- Encode depth values of the tile as offsets from reference values
  - Using a fixed number of bits per pixel



Morein & Natale [04]



Ornstein et. al [05]

Offset Compression

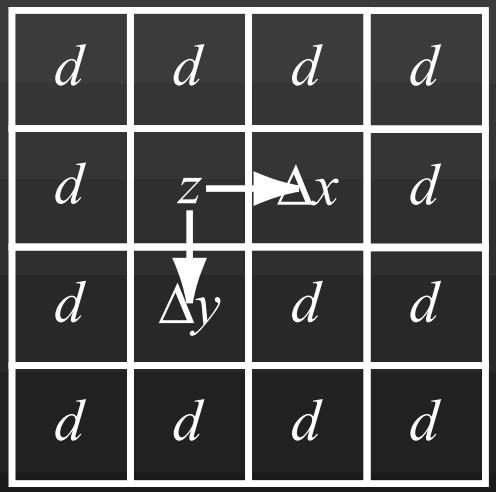
- Advantages
  - Excellent compression frequency
    - #compressed tiles / #total tiles
  - Robust to tessellation
- Disadvantages
  - Low compression ratios
    - Typically 3:2

Plane Compression

- Compute and store reference planes
  - Typically one or two planes
  - Represented as a point (depth value) and two deltas (screen space x,y)
- Store the depth value of each pixel as an offset to the reference plane
  - 0-5 bits to encode the offset
  - Helps when z is interpolated with high precision

Anchor Encoding

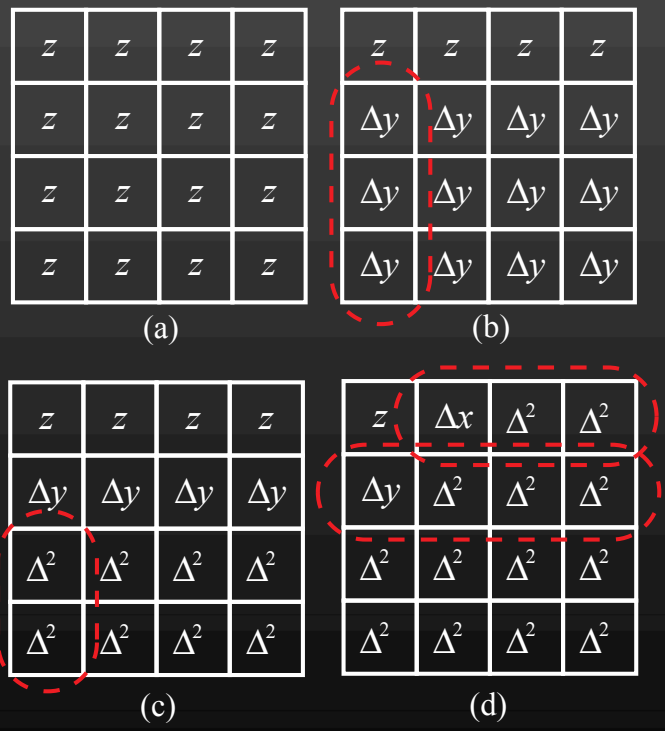
- Van Dyke and Margeson [05]
  - Compute a prediction plane based on 3 fixed points
  - For the remaining points
    - Store 5 bit correction offsets
  - 4x4 tiles, 3:1 compression



- Simple, offset robustness
- Low effective compression ratio

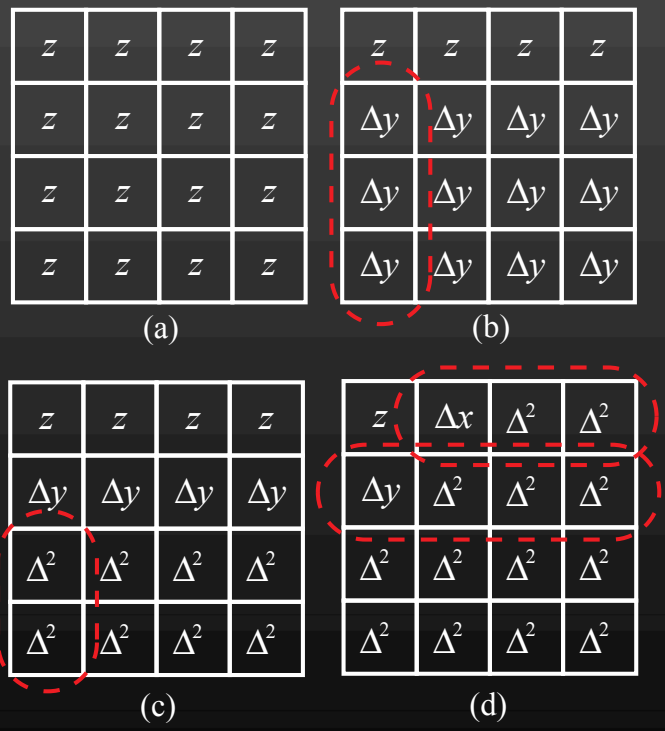
DDPCM

- Deroo et. al [02]
  - Compute 2nd order x,y differences
  - Target is planes
    - Second order difference of a plane is 0
  - Store the representation in (d), 2 bits per offset
    - Enough to cover variations due to high precision interpolation [-1,1]



DDPCM

- Deroo et. al [02]
  - 8x8 tiles, 8:1 compression
    - High compression ratio
    - Can handle some cases of two planes
    - Designed for big tiles



Plane Encoding Summary

- Advantages
  - Higher compression ratios than offset compression
- Disadvantages
  - Does not handle high tessellation as well as offset compression
    - Especially not for large tiles

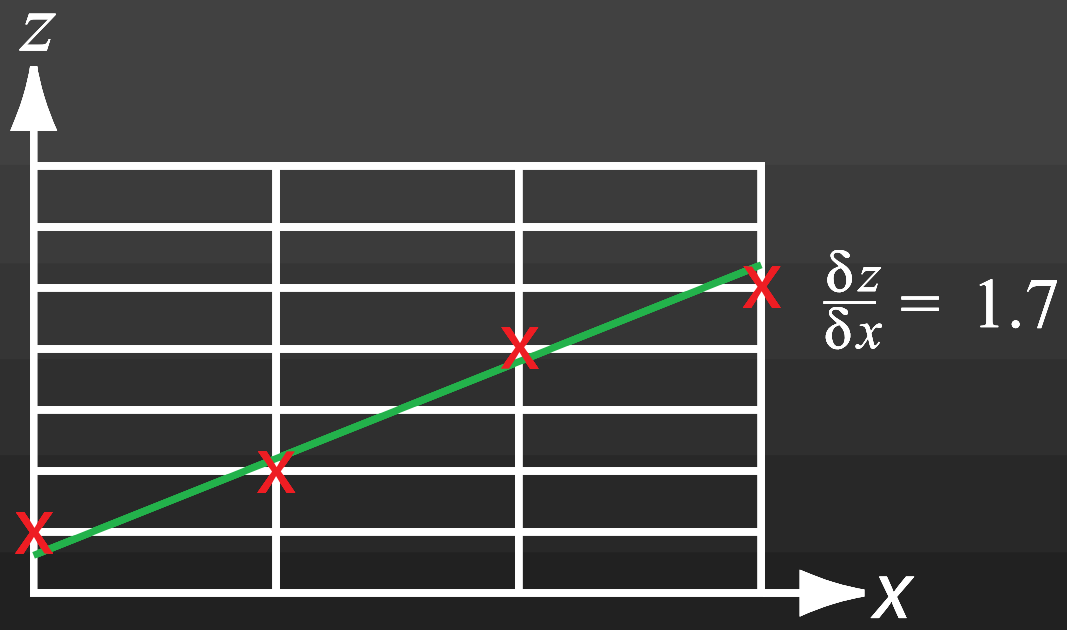
Our Algorithm

- Survey of efficient depth buffering
- A new depth compression algorithm

Our Algorithm

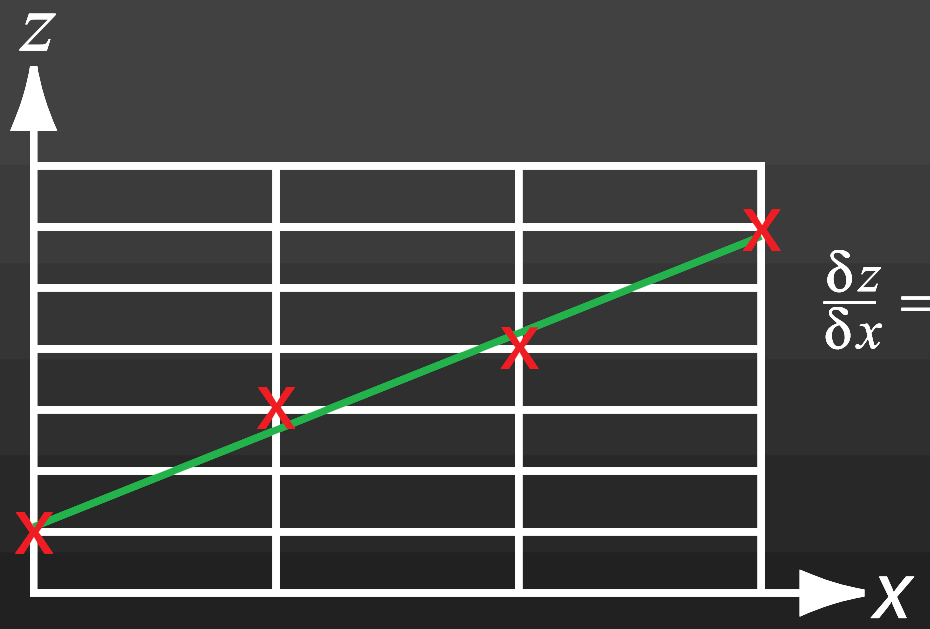
- Based on Bresenham's interpolation algorithm
  - Fixed integer increment
  - Plus a correction term (0 or 1)
- Find increment (plane delta)
- Store correction term using 1 bpp

Our Algorithm



z	1	2	4	5	$p = 1$
$\frac{\Delta z}{\Delta x}$	1	2	1		$\frac{\Delta z}{\Delta x} = 1$
Δ^2	0	1	0		

Our Algorithm



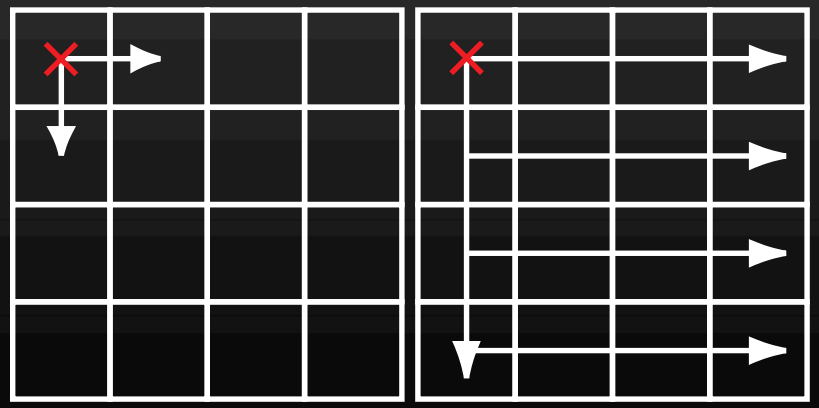
z	1	3	4	6
$\frac{\Delta z}{\Delta x}$	2	1	2	
Δ^2	0	-1	0	
Δ^2	1	0	1	

$p = 1$

$\frac{\Delta z}{\Delta x} = 1$

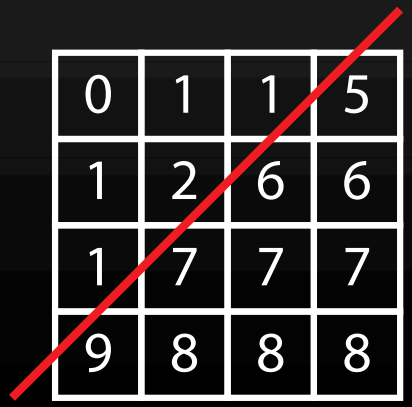
Our Algorithm

- Generalizes to two dimensions
  - Fixed reference points / deltas
  - Fixed traversal pattern



Two Plane Algorithm

- Compress a tile where two planes are separated by a single edge
- Done by doing repeated executions of the original one-plane algorithm.
  - Compute reference plane for each corner
  - Discard identical planes
  - Merge compressed results

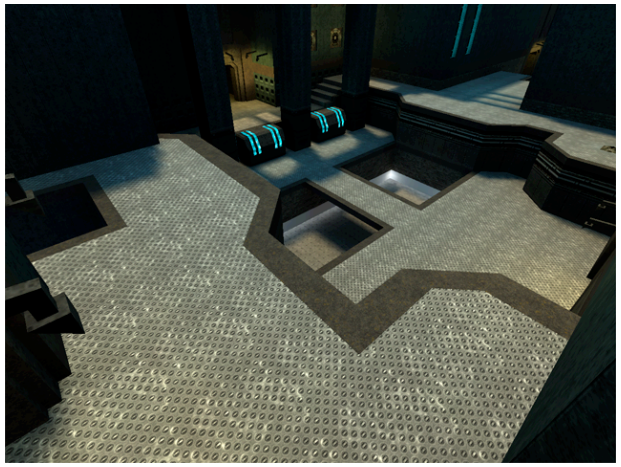


Summary of Our Algorithm

- Similar to previous plane encoders
- 1 bit per pixel offsets
  - Can compress >99.9% of the tiles that are compressible with DDPCM
  - Advantageous when compressing small (4x4) tiles
  - Handhelds, mobile phones etc.

Evaluation

Game Scene 1

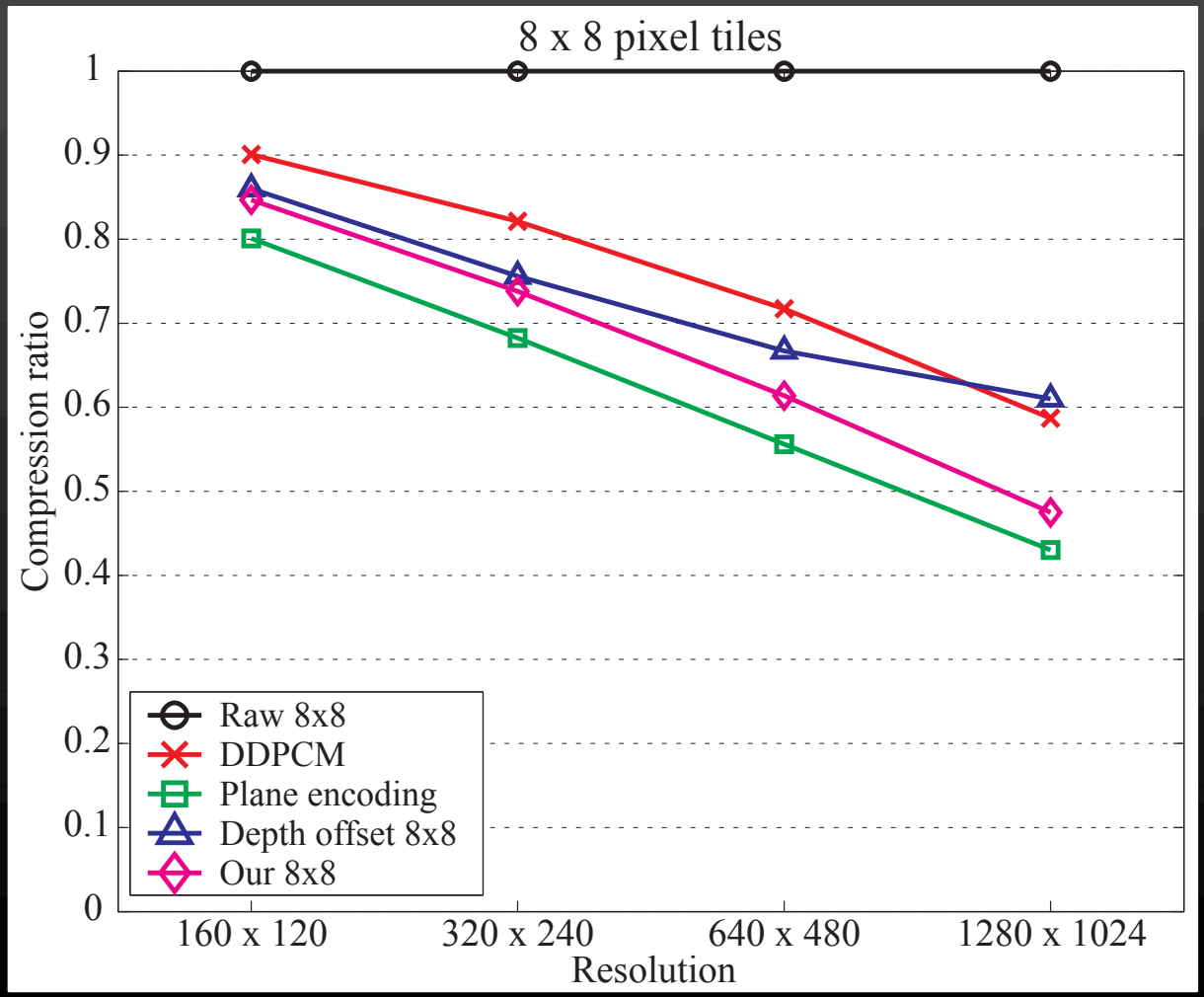


Game Scene 2

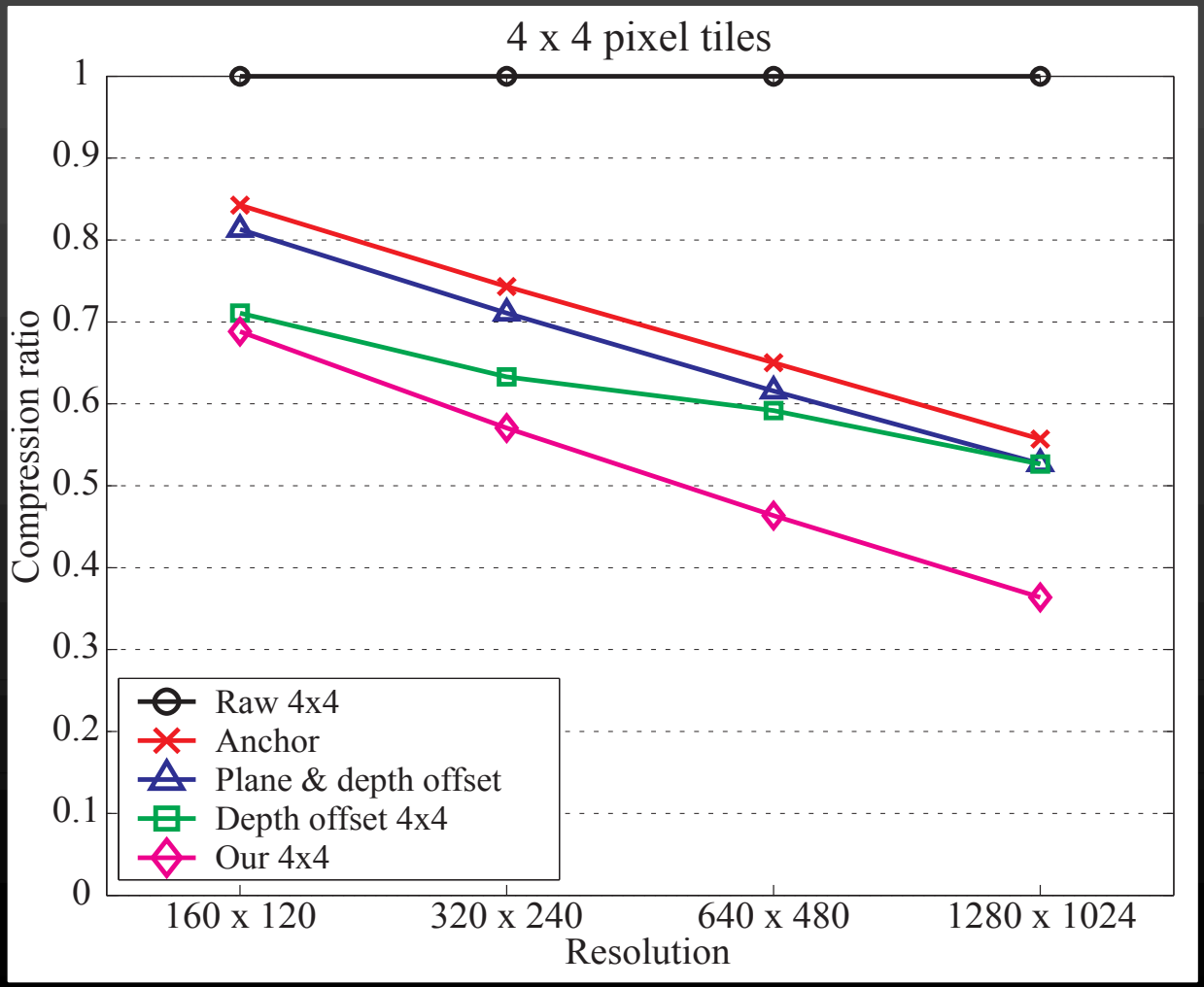
Sponza

- Rendered at different resolutions
  - Simulate varying tesselation
  - Avg. triangle area: 0.6 - 600 pixels

Evaluation


8 x 8 pixel tiles
Compression ratio
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
160 x 120
320 x 240
640 x 480
1280 x 1024
Resolution
Raw 8x8
DDPCM
Plane encoding
Depth offset 8x8
Our 8x8

Evaluation


4 x 4 pixel tiles
Compression ratio
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Raw 4x4
Anchor
Plane & depth offset
Depth offset 4x4
Our 4x4
160 x 120
320 x 240
640 x 480
1280 x 1024
Resolution

Summary and Future Work

- Overview of hardware depth buffering
- New compression algorithm
  - 1 bpp “for free”
- Future work: Multisample depth compression

Thank You

- Jukka Arvo
- Petri Nordlund
- Vetenskapsrådet
- Swedish foundation for strategic research (Mobile Graphics grant)
- NVIDIA fellowship

Questions?