Imagination

HPG 2023

IMG LABS : GPU RESEARCH

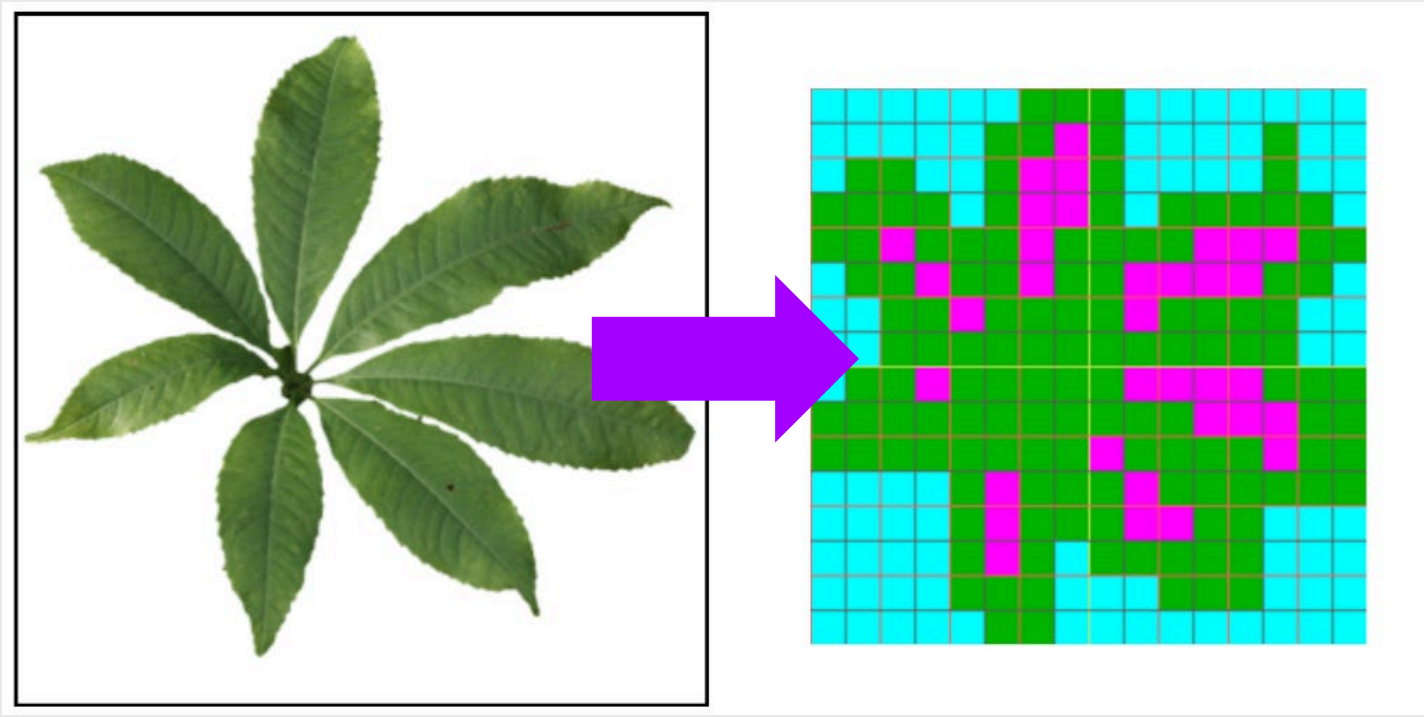
COMPRESSED OPACITY MAPS FOR RAY TRACING

June 2023

Simon Fenney & Alper Ozkan

OPACITY MAP/MASK OVERVIEW

- The problem are we addressing?
- What are opacity maps & how do they help?
- Why would we want to compress them, and how?



Q: WHAT IS THE ISSUE?

A: ALPHA TESTS

Particularly if you have *lots* of alpha tests

Complex Geometry in Ray Tracing

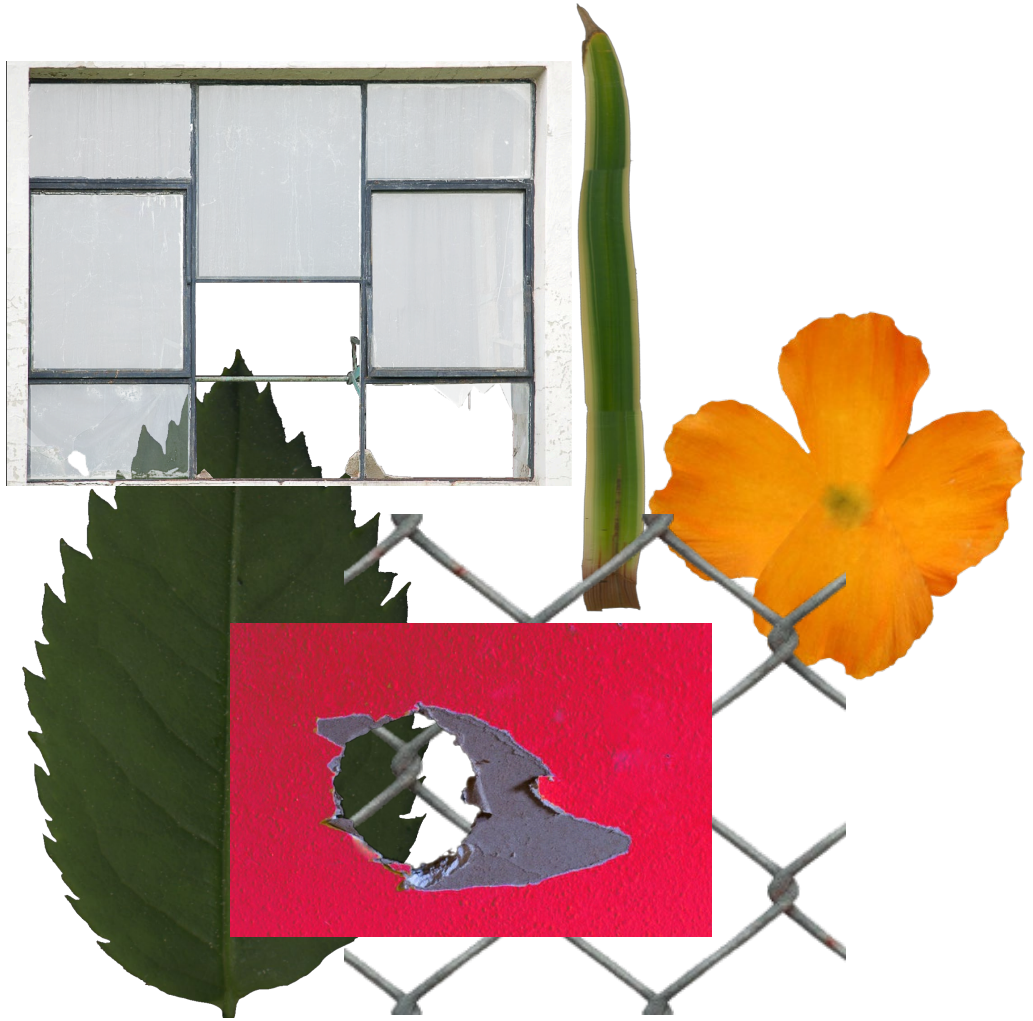
Triangles are great until …

Artists/Designers want to model shapes with complex outlines e.g.

- ‘Flora’ - leaves / petals / billboards of entire trees
- Building components: grills, windows, mesh fences
- Damage effects

Modelling purely with ‘solid’ triangles would be costly.

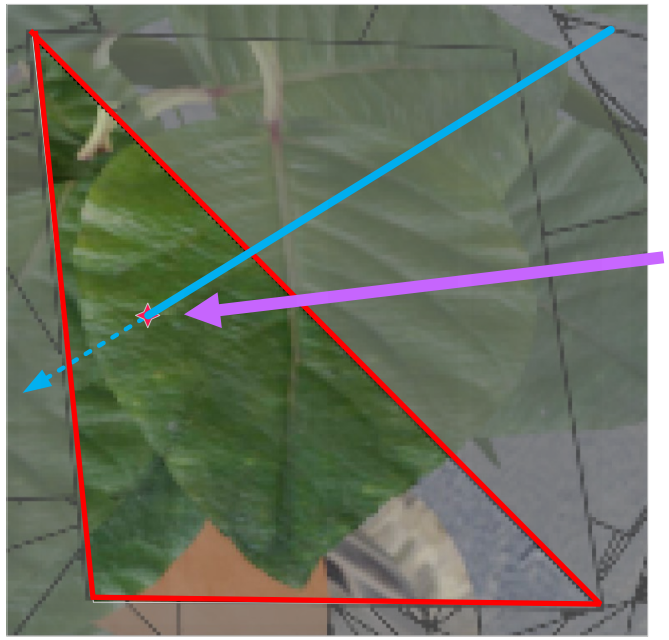
Solution (like rasterisation) → Use ‘alpha testing’



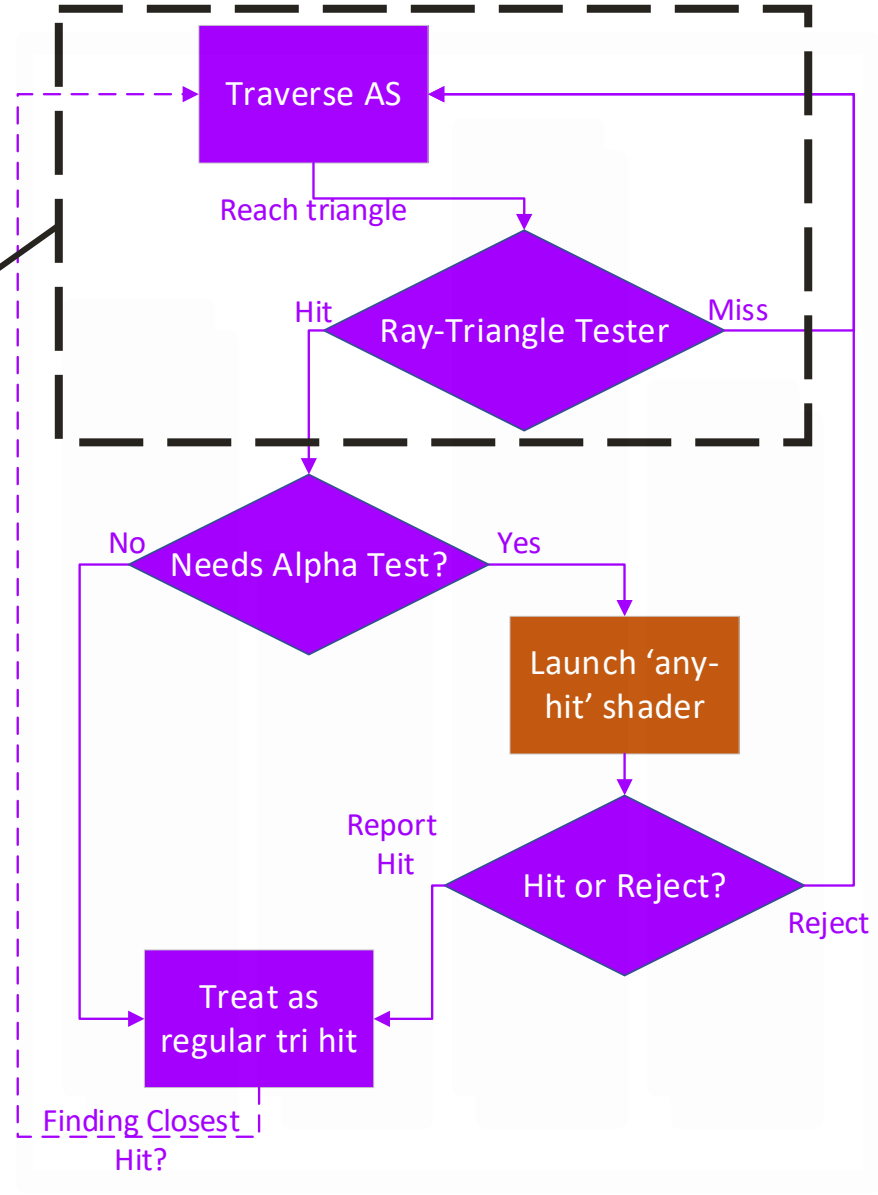
API Support

Vulkan and DirectX ‘Any-Hit shaders’

WHEN the **A**cceleration **S**tructure **T**raversal (AST) finds a ray hits a triangle:



Ray-Triangle intersection found.
Do we really want to keep it?



Alpha testing with ‘Any-Hit shaders’ : 2

TL/DR: Much more expensive than ‘opaque’ triangles.

Very flexible

- Via textures *or* computed e.g., code a ‘circle’

But interrupting the **A**cceleration **S**tructure **T**raversal (**AST**) incurs a penalty

Memory is *s l o w* …

May need to hide **many** 100s of clock cycles of latency.

- schedule any hit shader
- Have potential texture cache miss
- restart **AST** for ray

There may be many layers of alpha-testing, e.g., San Miguel and PBRT “Landscape”

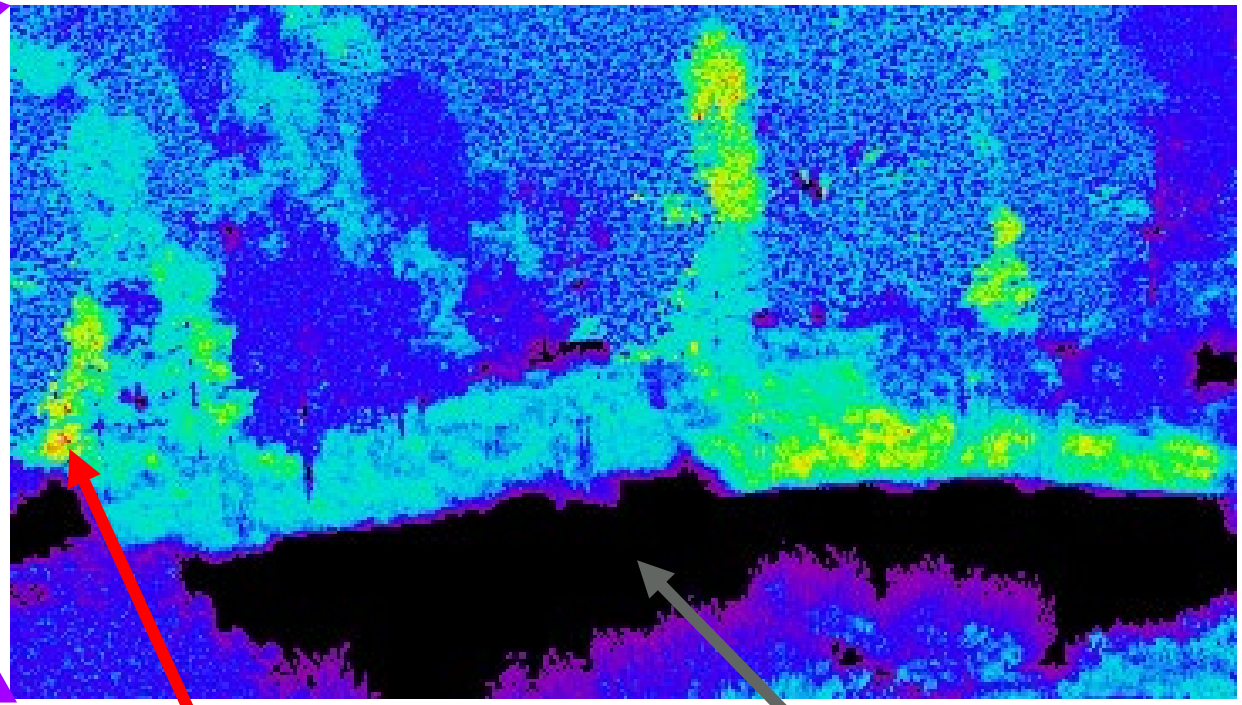
Example…

"Physically Based Rendering" Landscape Scene

Consider just the primary rays

A nice shrubbery may be too expensive.

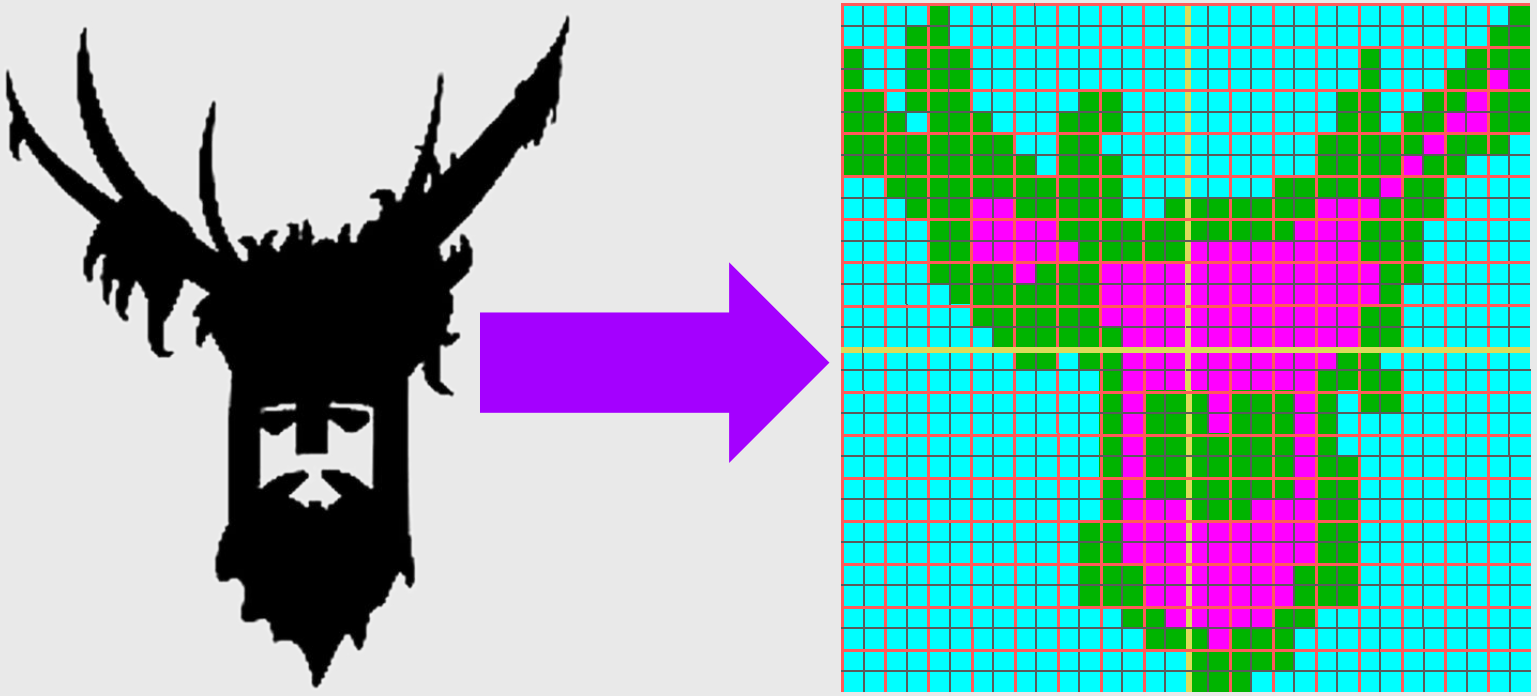
Average Alpha Tests/ray



Significant overhead 😢

'Opaque' primitives 😐

DO LESS WORK VIA OPACITY MAPS / MASKS



Opacity Masks: Background

Presented at HPG 2020

In 2020 Gruen, Benthin and Woop presented opacity masks

- An 'alpha tested' triangle has N^2 ***precomputed*** sub-triangles

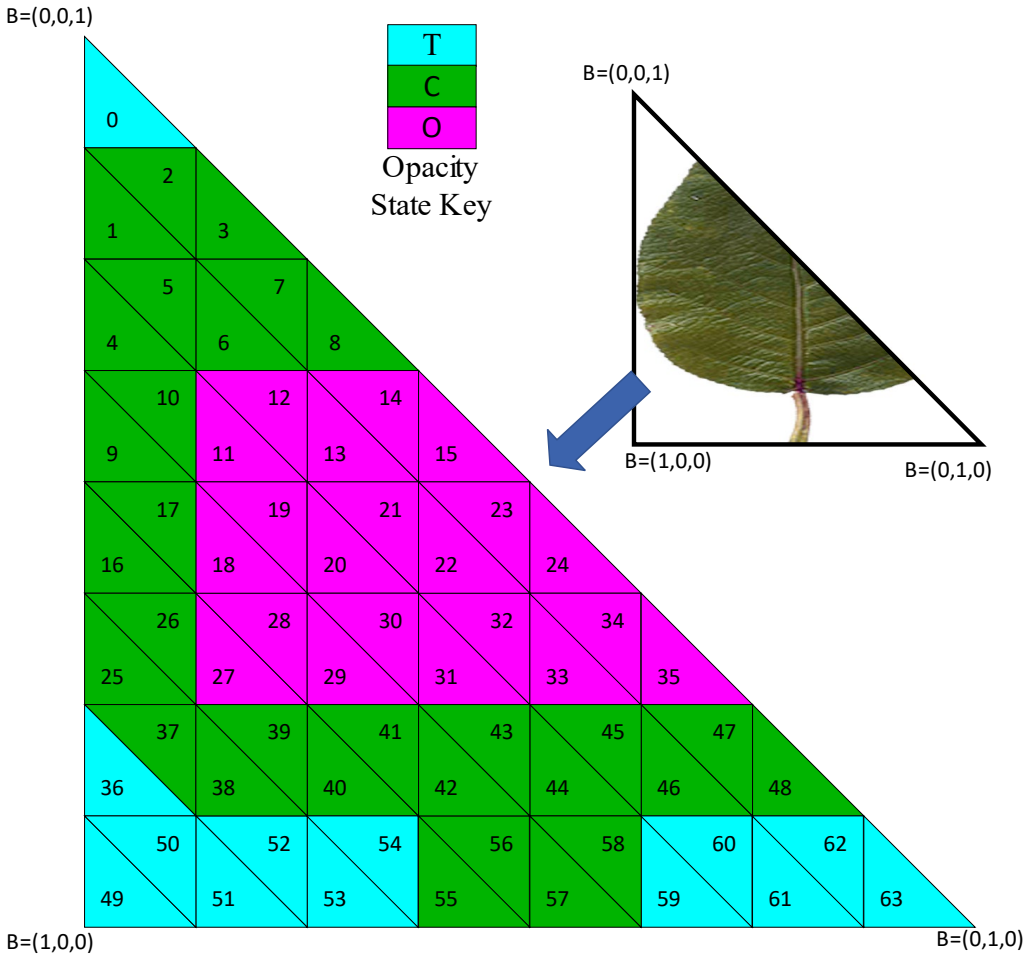
Each sub-triangle has one of 3 states:

- **T**: Completely Transparent (cyan)
- **O**: Completely Opaque (magenta)
- **C**: Check using 'any hit shader' (green)

T and **O** states entirely avoid launching the any-hit shader

The mask can be stored with the triangle/acceleration structure, and Ray-Triangle tester reads the mask.

This example gives a 53% saving – can we do better?



Opacity Mask/Map resolution VS ‘any-hit’ reduction

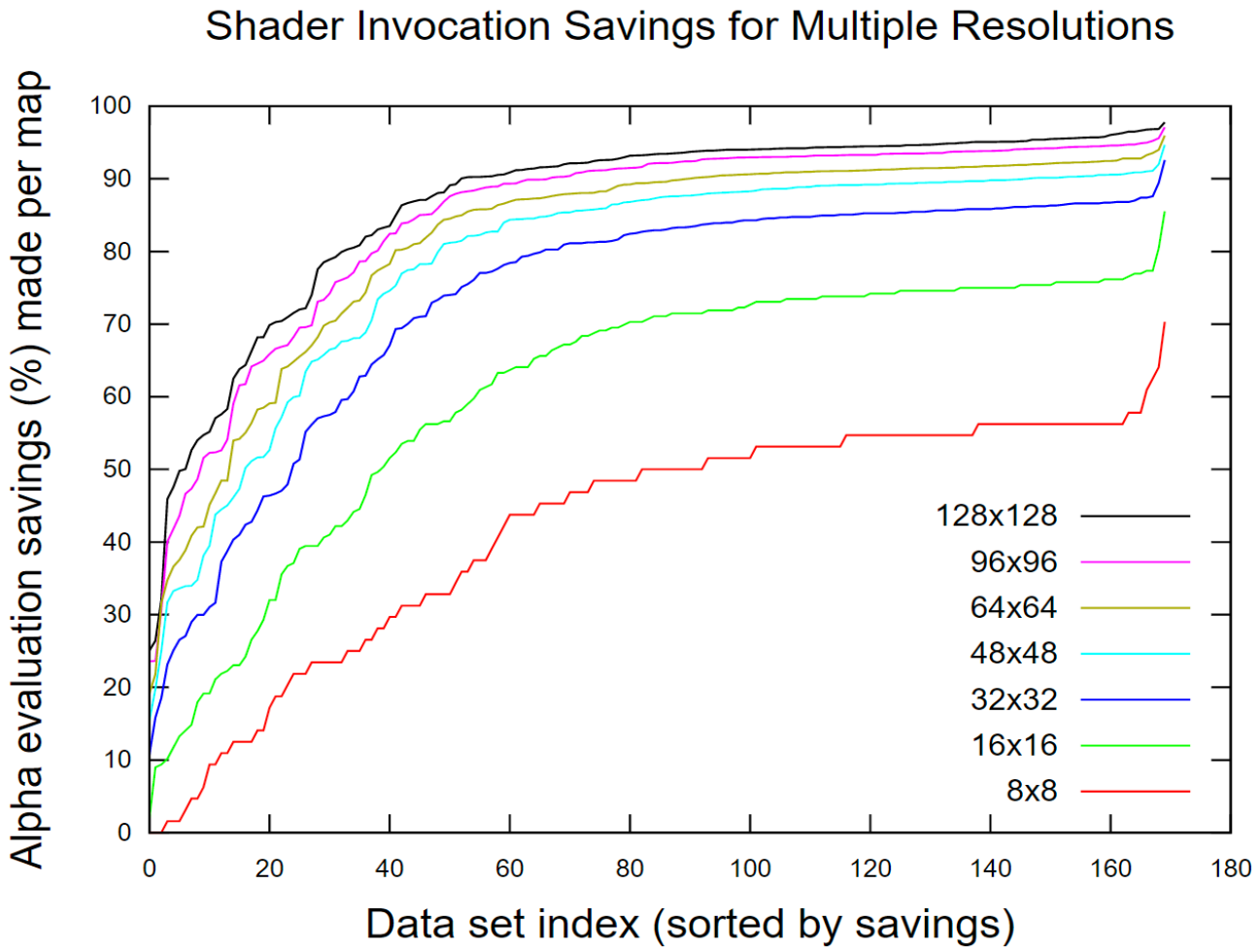
Higher resolution ⇒ greater savings

Gruen et al used a set of 4 alpha textures to show that Higher Resolutions ⇒ Greater Savings.

We found similar with a set of 170 textures…

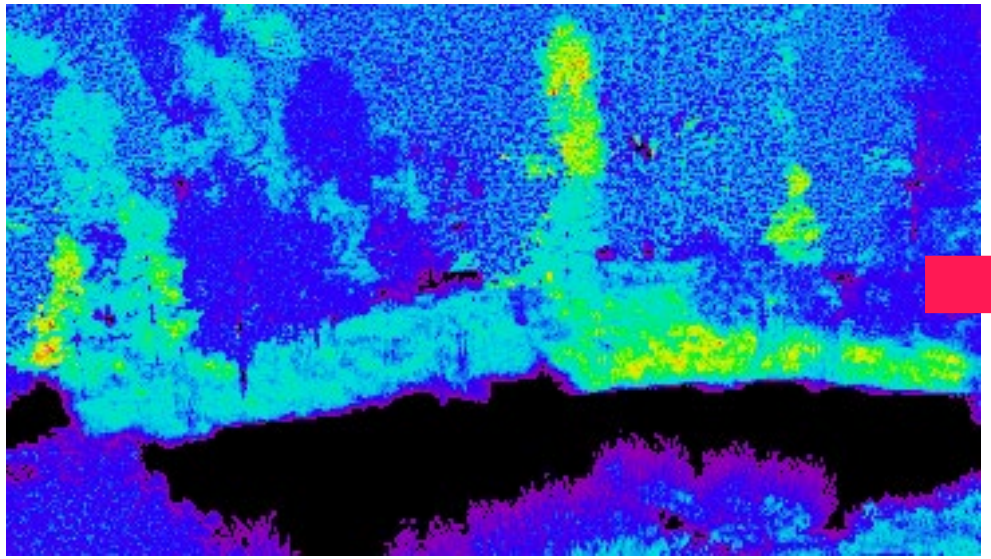
… though the gains diminish.

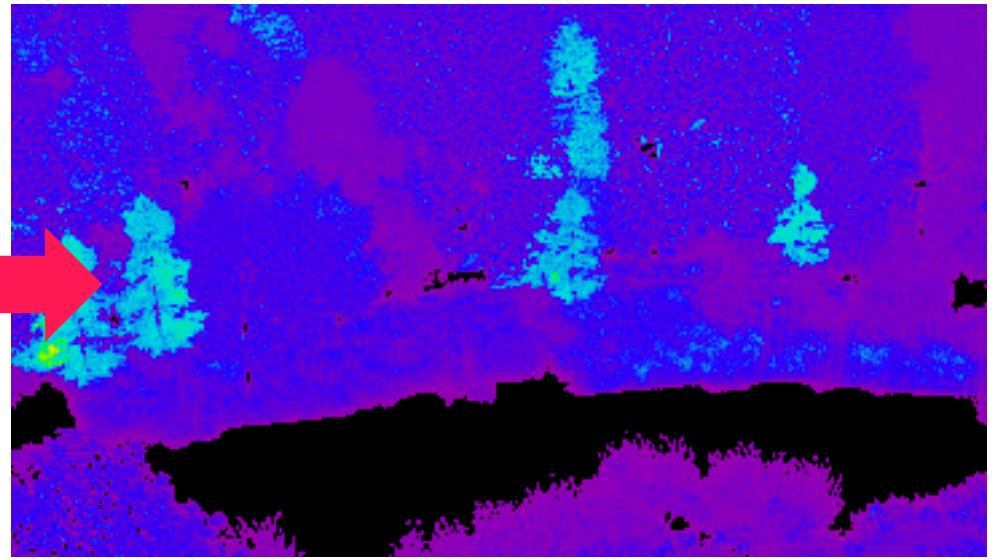
But it certainly helps in alpha tested scenes..

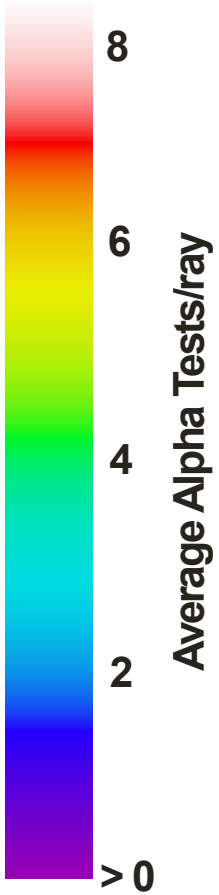


PBRT Landscape revisited with opacity maps

Alpha tests performed for just **primary** rays: Heat map







Original

With 'opacity maps'

CAN WE DO BETTER?
SOME OBSERVATIONS

Why do individual triangles?

Can we do better than 2bits/region?

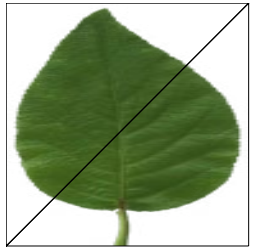
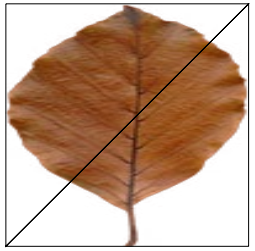
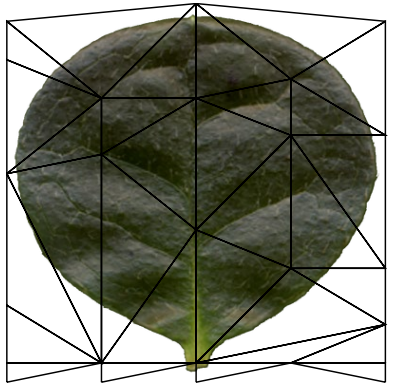
Assorted Observations: 1

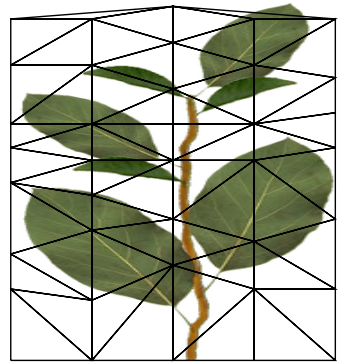
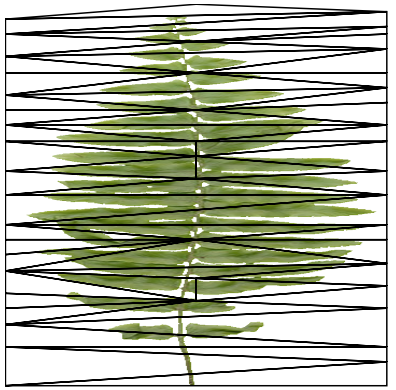
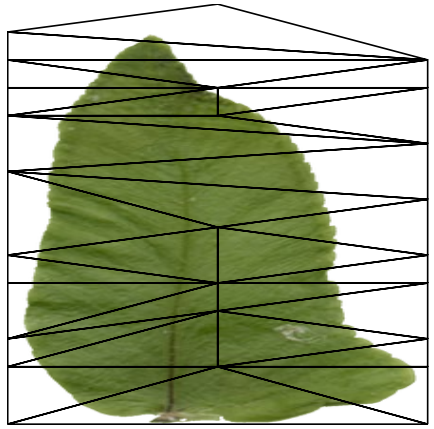
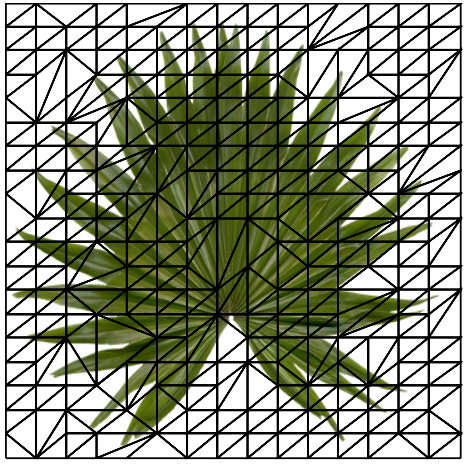
Earlier works define a map per triangle

We started with individual triangles but…

Flora models commonly use meshes of at least 2 triangles

Can we exploit coherency?

Assorted Observations: 2

The underlying hardware ‘primitive’ *might* be pairs of triangles

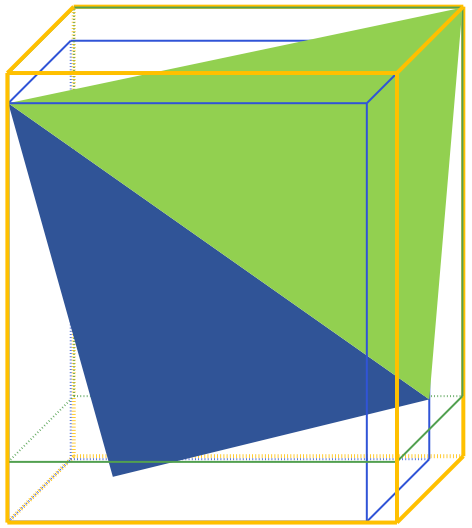
Some hardware systems (e.g., Intel, Imagination/PowerVR) store and/or test pairs of adjacent triangles (i.e., with a shared edge)

Why do this?
(See the HPG2023 poster “Fast Triangle Pairing for Ray Tracing”, Aman & Fenney)

1. **The AABB for a *pair* of triangles often not much bigger than the child boxes. Can reduce *average* number of AABB tests.**

2. **Testing a ray against a pair of triangles shares maths of the common edge**

3. **≈ 33% reduction in vertex storage vs individual triangles**



Assorted Observations: 3

Storage Costs

Opacity mask –stored as 2 bits/state

Gruen et al noted could be *"compressed into less than 2 bits per sub-triangle"*

We read as $\log_2 3$ bits per sub-tri

- Want random access – perhaps 5 sub-tris per byte? (close to optimal)
- Allows completely arbitrary content…

…But textures are not 'noise' so a smaller footprint seems feasible

Want higher res maps → larger savings (but comes at a cost)

So compression desirable but need *fast random access*

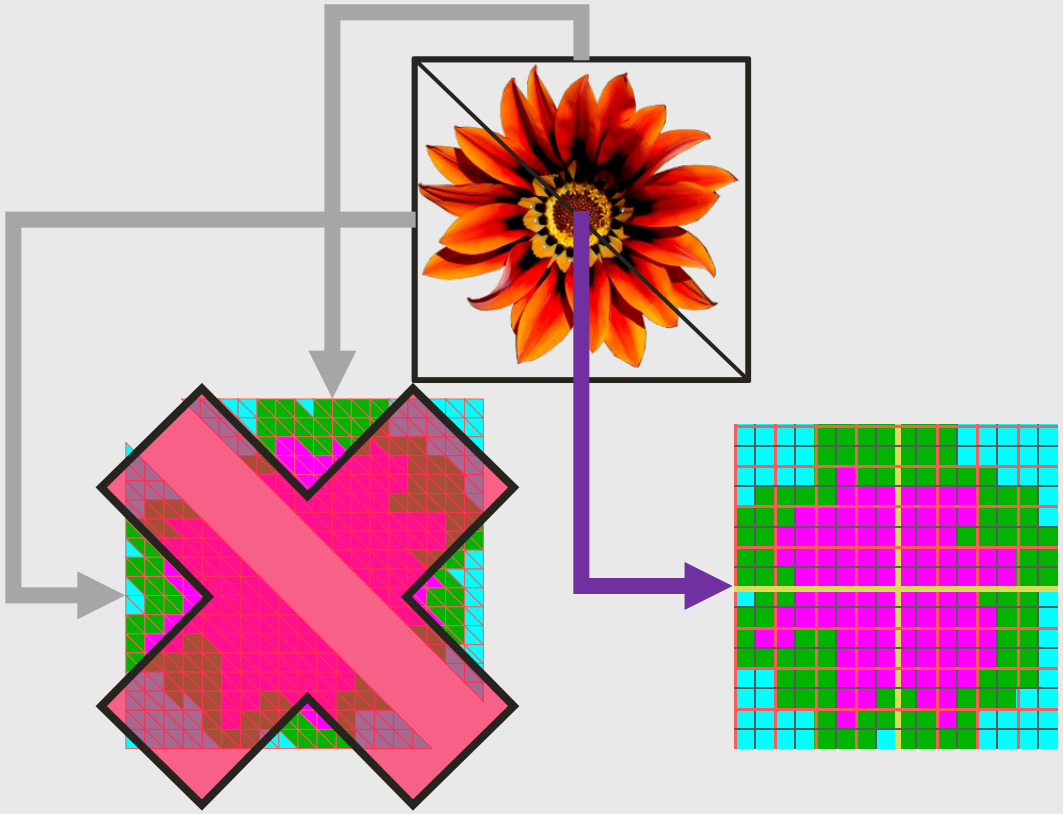
~~Huffman, RLE, Arithmetic Encoding~~

Want to store Opacity Map together with vertex XYZs for efficiency

- Avoid additional latency
- Probably should be ≈ same size as vertex data

OUR APPROACH

- Use *square* opacity maps (with square sub-regions) for *pairs* of triangles
- Resurrect Vector Quantisation (VQ) for compression in graphics



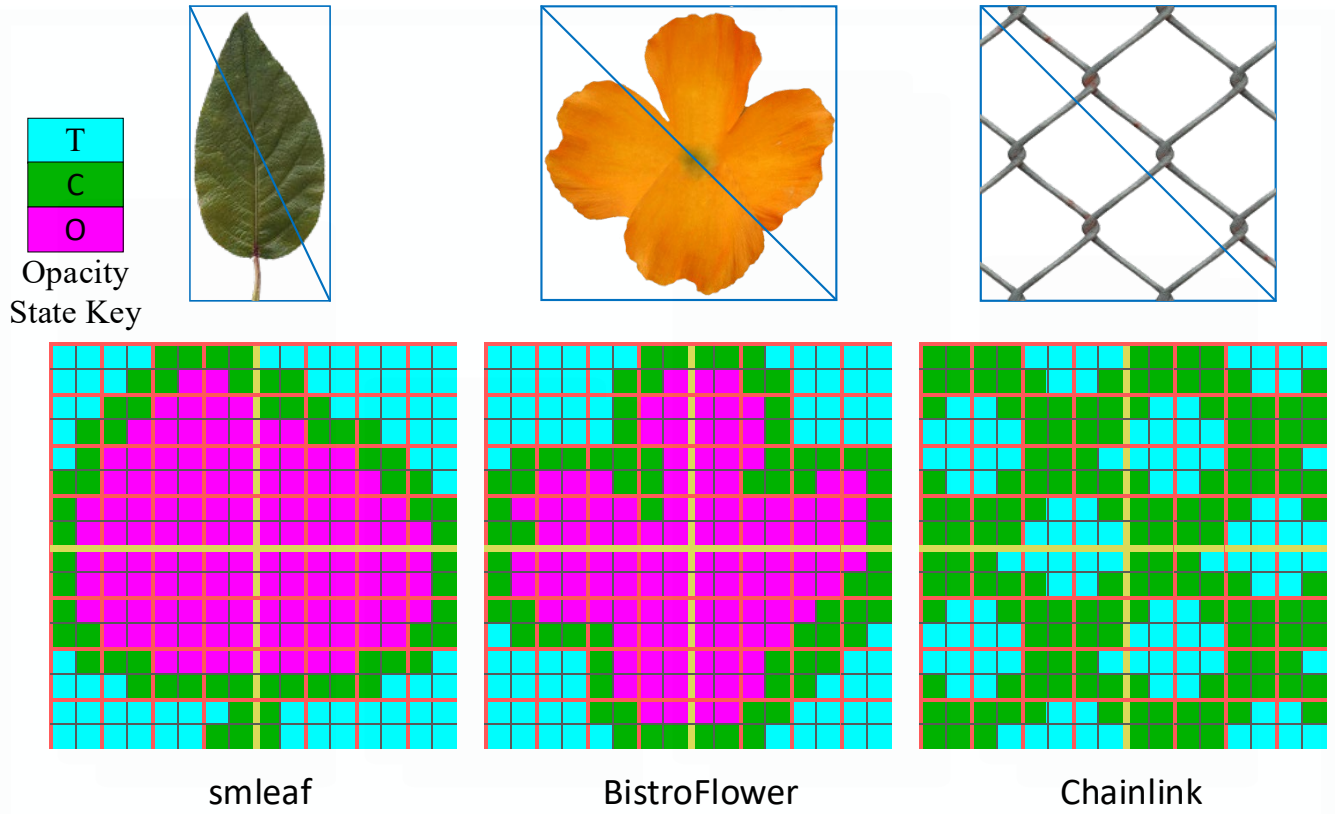
Square Opacity Maps

Illustrating with 16x16 resolution

Examples of textures mapped to two triangles

We expect

- **Similar behaviour across the shared edge**
- **To be frequency limited → no instant transitions from T to O.**
  - Referred to as “continuity assumption” in paper



Compression/Decompression

Need Fast, Cheap and “Fairly Good”.

***Decompression* needs to:**

- **Be fast & provide random access**
- **Be cheap! (small HW & low power)**

***Compression* can be lossy**

- ***provided* the loss is conservative/safe, i.e.**

  ✓ O→C or T→C ✓

  ✘ C→O nor C→T ✘

- **But not *too many* lossy substitutions → ‘undo’ benefits.**

Resurrecting VQ

"he's not dead, he's, he's restin' '"

VQ used in Texture Compression in late 1990s

e.g.

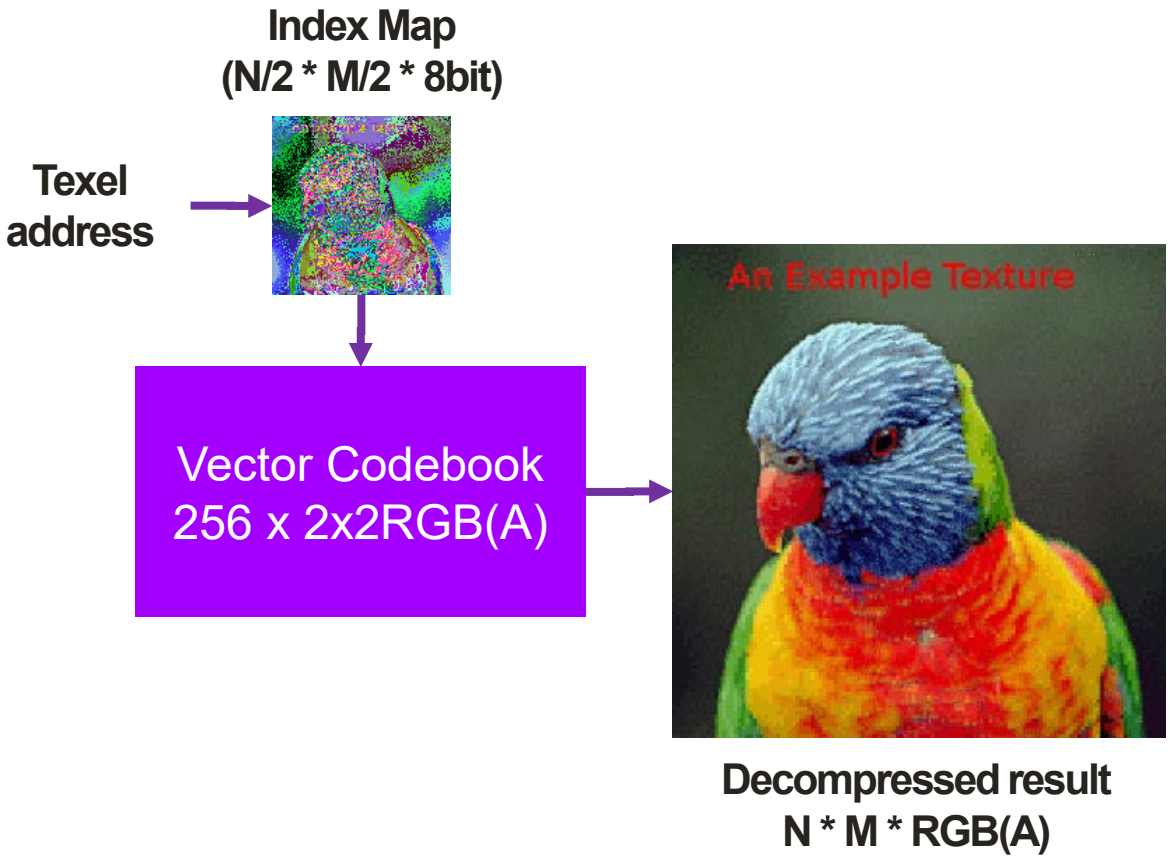
- "Rendering From Compressed Textures", 1996, Beers et al
- Sega Dreamcast, 1998, ≈2bpp, ≈1bpp
- (and even earlier if you count palette images)

Is a *lossy* compression scheme:

- Pigeonhole principle says so!

Simple decompression algorithm (e.g., 2x2 vectors)

- Fetch index → Access Codebook (LUT) → get texel from vector



So why did VQ disappear?

It had high compression with reasonable quality…

It fell out of fashion

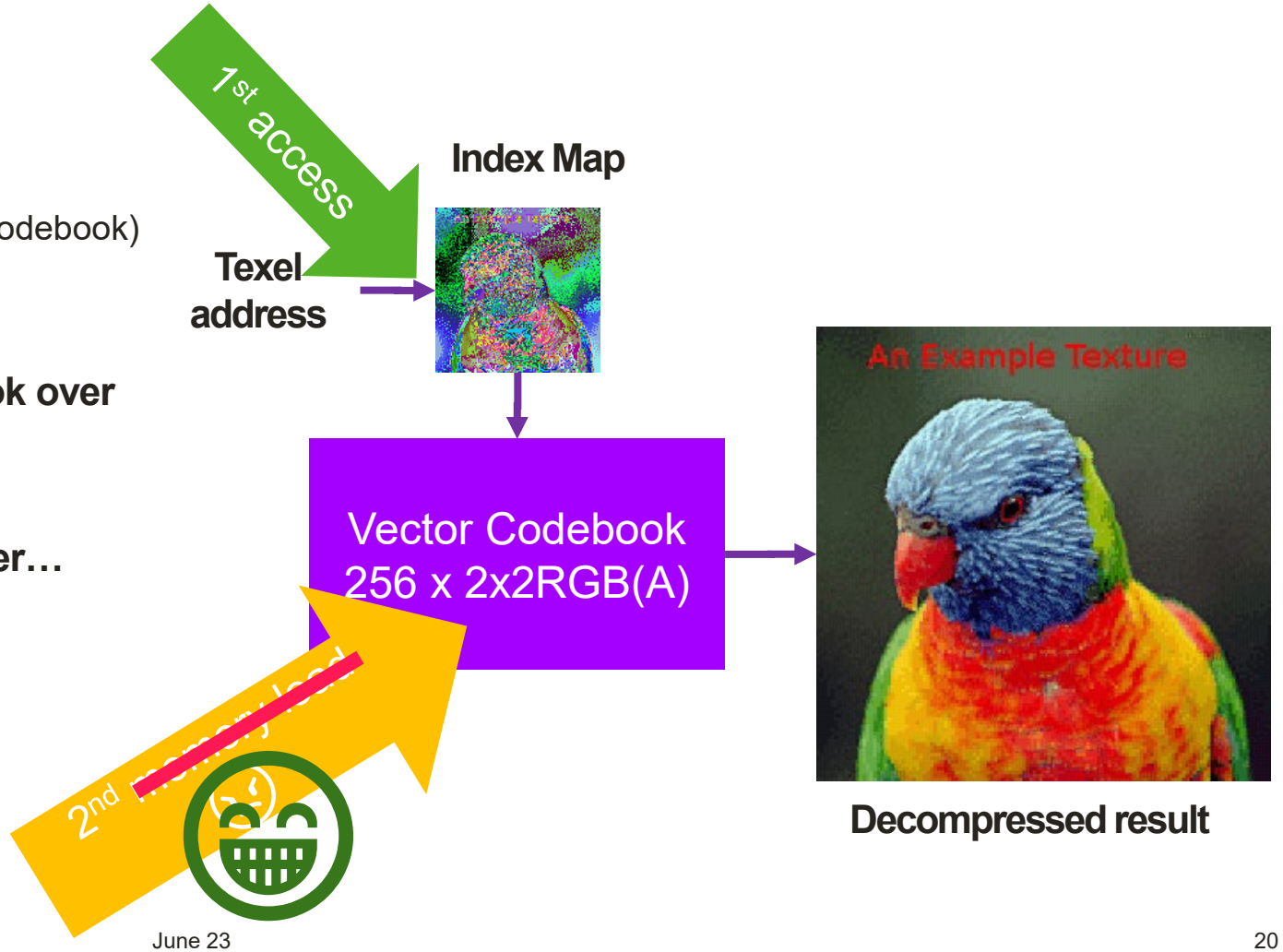
- Cost of hiding memory latency of codebook access
  (e.g., Dreamcast had a 2nd texture cache purely for the codebook)
- Compresses texture globally (not an issue here)

Other texture schemes, e.g. S3TC, PVRTC, ETC, took over

- no second, dependent memory access.

***But* if the index map & codebook are loaded together…**

…the latency of the 2nd access can be eliminated



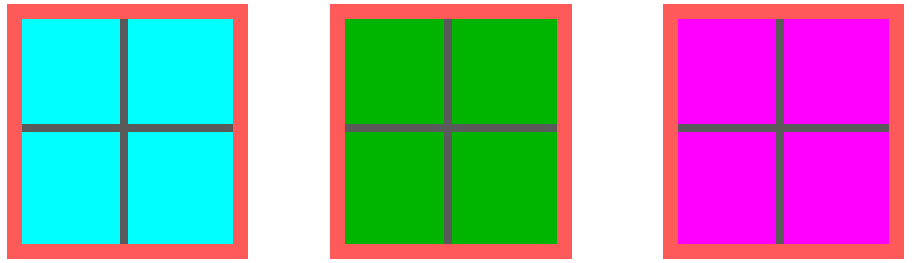
Initial investigation: VQ2: 2x2 vectors

A modest first experiment

Target was

- 1bit/region
- Use a fixed storage, e.g., 256 bits
  ≈ ½ a typical cache line or slightly less than XYZ vertex data (4x3 floats)

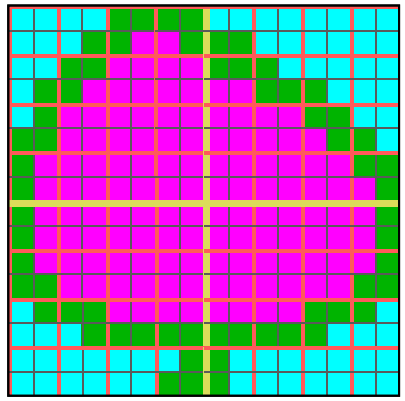
→ Forces 16x16 resolution.



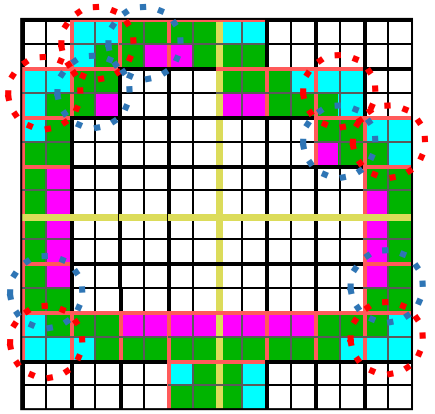
More observations from test data:

- **Uniform 2x2 vectors are common: *each* about 11 ~ 29%**
- **Often found ‘symmetry’ across quadrants:**
  - San Miguel leaf: Has instances of rotational symmetry
  - Other data had, e.g., reflections about X, Y, X=Y, X=-Y

Can use these to reduce size of stored codebook.



smleaf



Uniform vectors ‘removed’

Representing 2x2 vectors in a codebook

We will need to store vectors in a codebook.

Might appear there are $3^4 = 81$ different 2x2 opacity map vectors

- **Do we need 7 bits?**

By the 'continuity assumption', we can't have both O & T states in the same 2x2 vector

- **Only 31 possibilities → so can use (a virtually ideal) 5 bits**

Leads to an utterly trivial encoding

- 1 bit → palette mode: {C,T} or {C,O}
- 4 x 1-bit indices

Handling Per Quadrant ‘Transformations’

Modifying the Top Right, Bottom Left and Bottom Right vectors

Three of the quadrants each have a ‘transformation matrix’

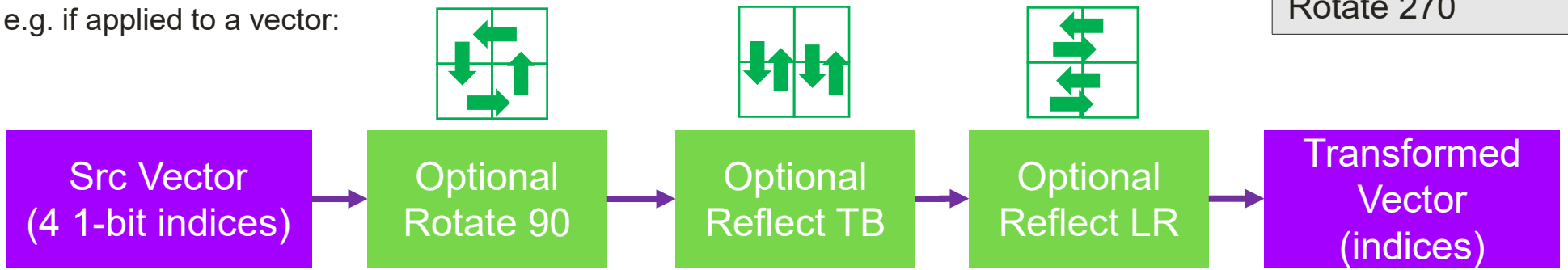
Transformations are chosen from a “palette” of 8

Each produced by chaining 3 optional, simple transforms

Can be applied either to stored vectors *or* coordinate LSBs

e.g. if applied to a vector:

Transformation	Encoding		
	Reflect LR	Reflect TB	Rotate 90
Identity	0	0	0
Rotate 90	0	0	1
Reflect Top/Bottom	0	1	0
Reflect X= -Y	0	1	1
Reflect Left/Right	1	0	0
Reflect X=Y	1	0	1
Rotate 180	1	1	0
Rotate 270	1	1	1



Fitting to the 256 bit budget

There are 64 vectors in the index map…

- 4 choices: 2 bits/index was too small,
- 16 choices: 4 bits/index was too big (leaves no space for anything else)
- 8 choices: is the “Goldilocks” value → 192 bits in total.

There are 3 transforms to store → 9 bits

…leaves 55 bits for the codebook → can store 11 vectors

Uniform vectors are *very* common, so don’t store in the codebook

- just use dedicated indices {0,1,2}

Stored Codebook is thus constructed as

- 3 “Global” vectors – indices {3,4,5}
- 4 x 2 Quadrant-specific vectors – indices {6,7}

A Naïve Compressor

Better Than It Had Any Right To Be

A very simple (AKA 'stupid') VQ compressor was written:

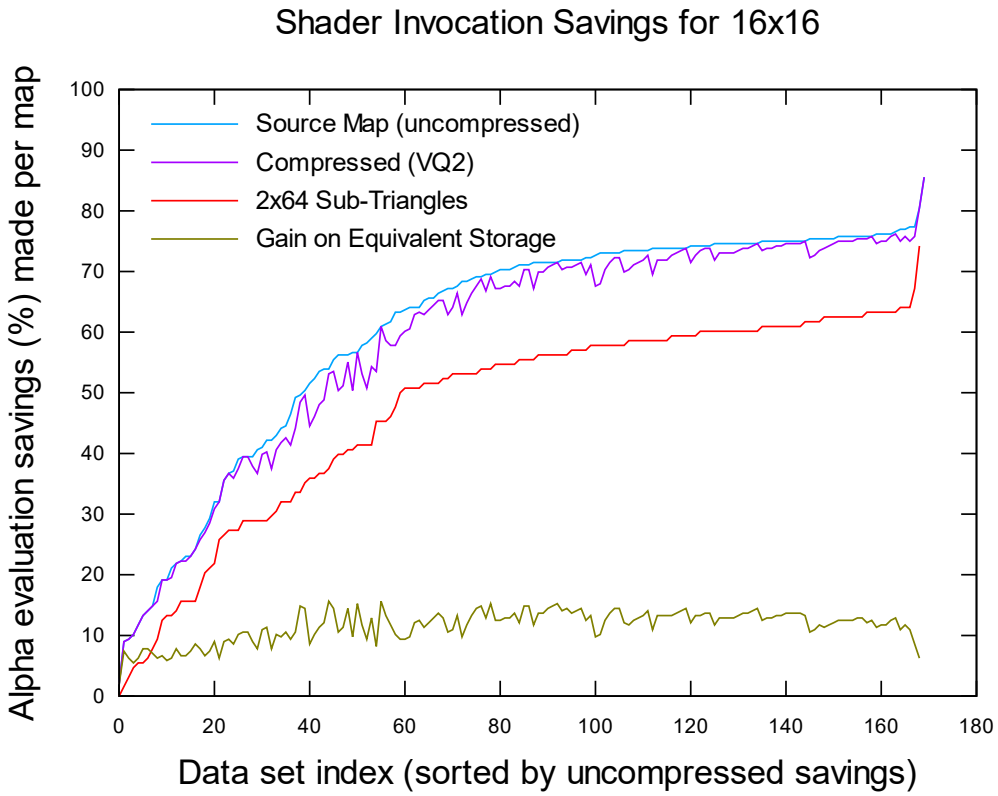
- **Try all 2^9 transformation combinations**
- **For given transform combination:**
  - While codebook not full
    - Greedily choose vector with fewest C states
  - Leftovers assigned to 'best compatible fit' in codebook

This worked much better than expected

- Very few conservative replacements made in practice (see graph)
- "Good enough" for a preliminary experiment

***Lossy* compression beat uncompressed 256-bit representation (2x 64 sub-tri opacity masks)**

≈ 8-16% gain



A MORE AMBITIOUS SCHEME: VQ4

Aim for higher resolution

without increasing storage *too* significantly

VQ4: 4x4 vectors

Can a higher resolution outweigh additional lossy compression?

New target

- ½ bit/region
- Fixed storage of 512 bits (≈1*ish* cache lines)

→Forces 32x32 resolution.

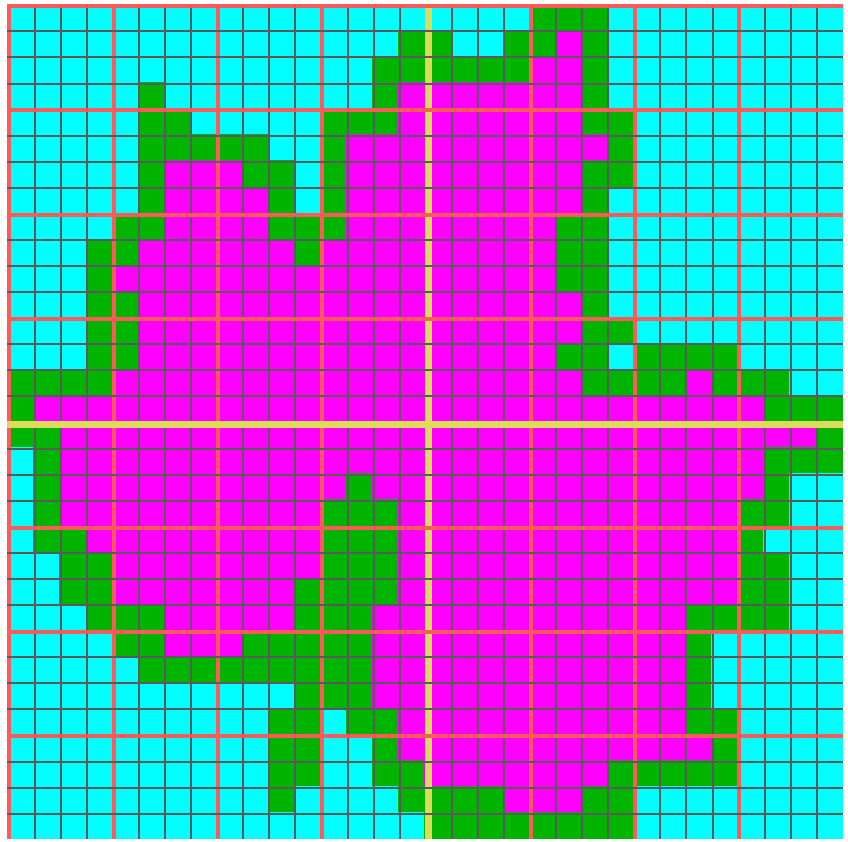
Uniform O & T vectors just as common.

- After all, it's just a higher resolution version of the previous 16x16

But expect fewer C states

Storage Budget target:

- 64 indices at 4 bits/index: 256 bits
- Quadrant transformations: 9 bits
- Budget for codebook → 247 bits

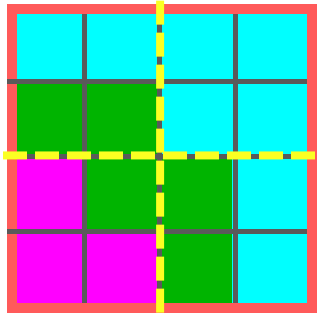


How to assign that codebook budget?

Enumerating cases…

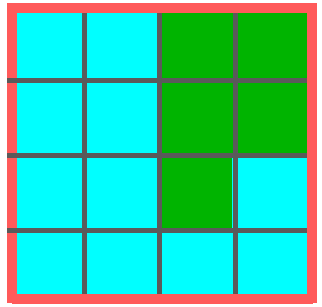
Unlike 2x2, a 4x4 vector *can* simultaneously contain O, T, and C.

- **Thankfully not 3^{16} (≈ 43M) possibilities…**
- **By continuity assumption, there are 'merely' 231713 (≈ $2^{17.8}$)**
  … but optimal encoding in 18 bits would be 'unpleasant'



Instead, store vector as four, VQ2 2x2s, i.e. 20bits.

- Slightly wasteful but simpler!

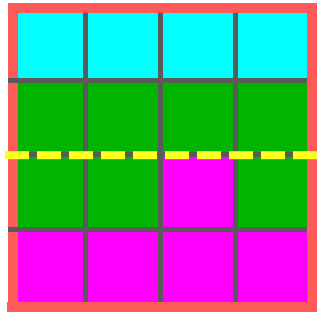


However, a number only had {O,C} or {T,C}…

→ Can use a 17-bit encoding similar to VQ2

Also, if we assume a horizontal (or vertical) partition of {O,C}, {T,C}

- Allows a slightly more general, 18-bit encoding.



Combine to meet 247 bit budget.

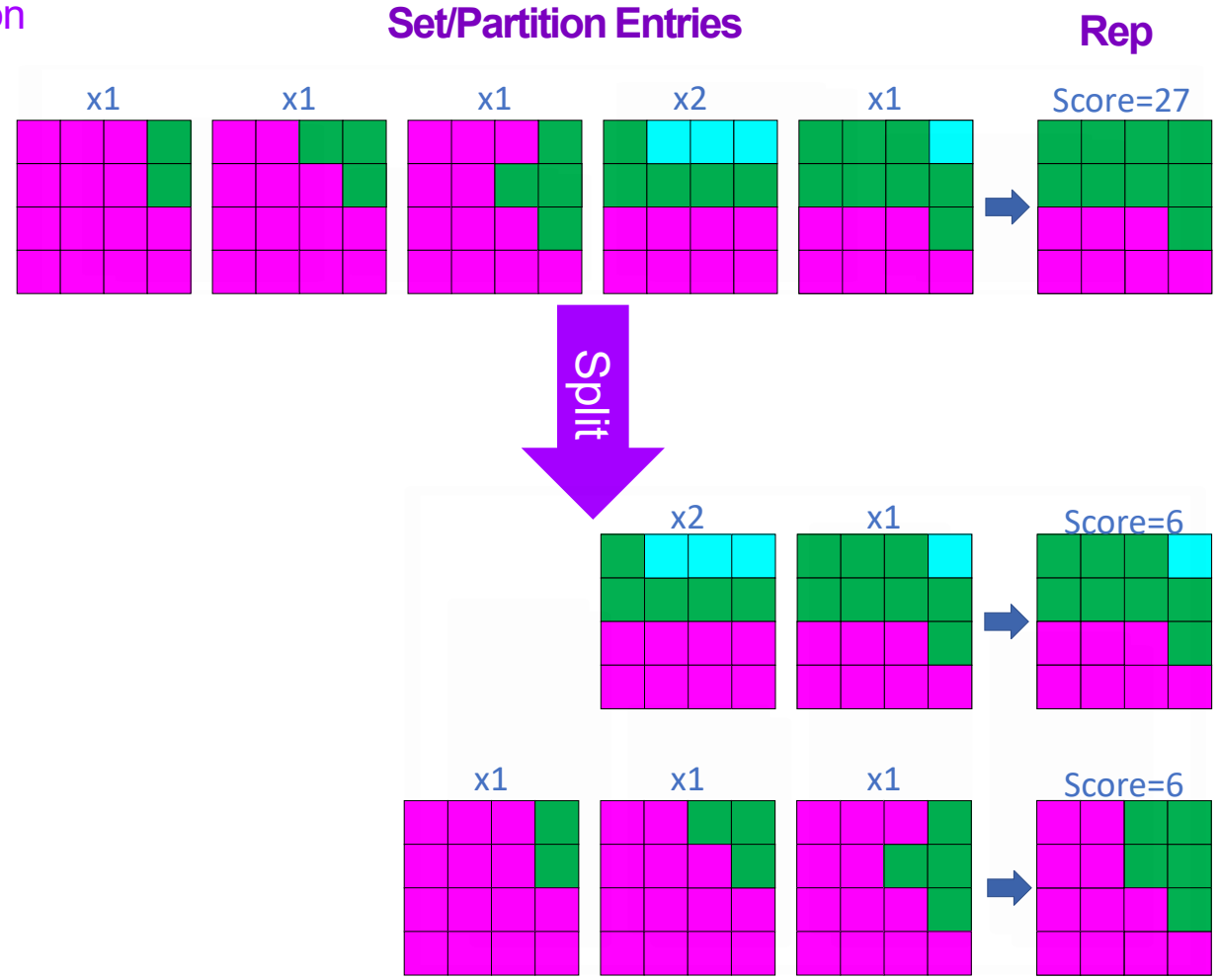
Needed a Better Compression Algorithm

Some similarities with Heckbert's "Median Cut" Colour Quantisation

Still a greedy algorithm but

- **Uses a top-down partitioning scheme**
- **Repeatedly subdivides a chosen set into 2 smaller sets**
- **…until have same number as codebook entries**

More details in the paper



VQ4 Compression Results

The **Big** Question: Does the additional resolution outweigh the loss of the scheme?

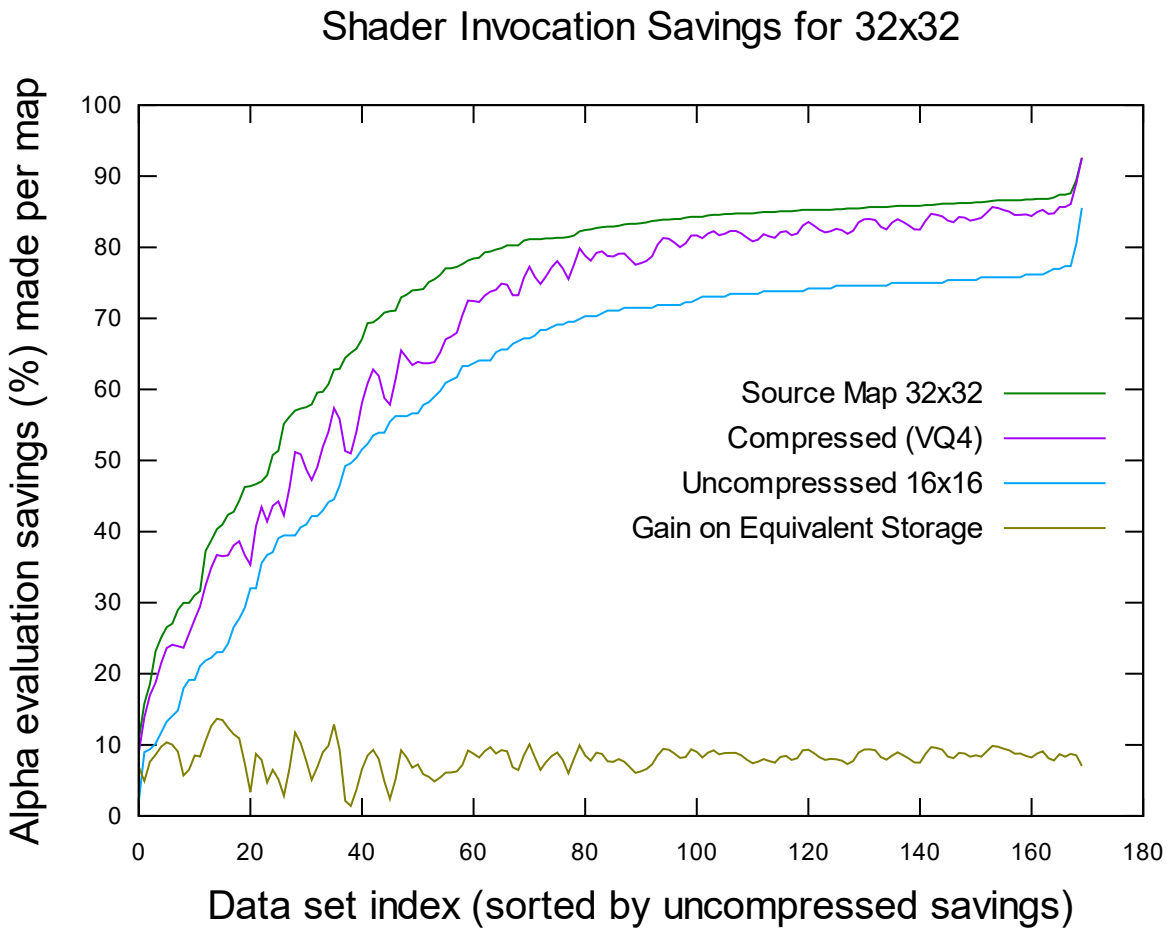
Retest using the same texture data, but at 32x32 resolution.

Though lossy

- **is still around 10% better than an equivalent uncompressed format (i.e. 16x16 at 2bit/region).**

Further, the additional hardware required to decompress the 32x32, over a raw 16x16, is expected to be tiny.

- negligible compared to rest of the ray-triangle tester!



Future work

Though the VQ4 compressor is more sophisticated than VQ2's …

- It is still a greedy algorithm
- A more intelligent search (e.g. backtracking) probably could do better.

Assuming implicit uniform 4x4 C vector is probably suboptimal.

- We would like to investigate ways of synthesising other vectors

Vulkan's *Opacity Micromap* has a way of mapping from multi-level to just O & T (e.g., for stochastic sampling cases)

- Investigate "hash/dither" to map from {O, C, T} to just {O,T}

Can we adapt the AS Traversal for 'shadow/AO' determination to defer handling C cases?

Conclusions/Summary

It was known that

- **Opacity Masks/Maps could lead to savings in ray tracing alpha-tested primitives and**
- **Higher resolutions give greater savings (up to a point)**

We investigated using fixed-rate/lossy compression for opacity maps

- **To allow higher resolutions for a given storage budget**

Have proposed using ‘square’ maps for *pairs* of triangles using VQ compression.

Demonstrated that, despite compression being lossy (i.e. conservative), it is beneficial

Imagination

THANK YOU FOR LISTENING

QUESTIONS?

A quick note on test data

An assortment of 170 images

Taken from various sources

- E.g., San Miguel, Sponza, Unreal Engine's 'Downtown', Lumberyard-Bistro, Unity, etc
- Obvious 'atlas' textures were manually subdivided

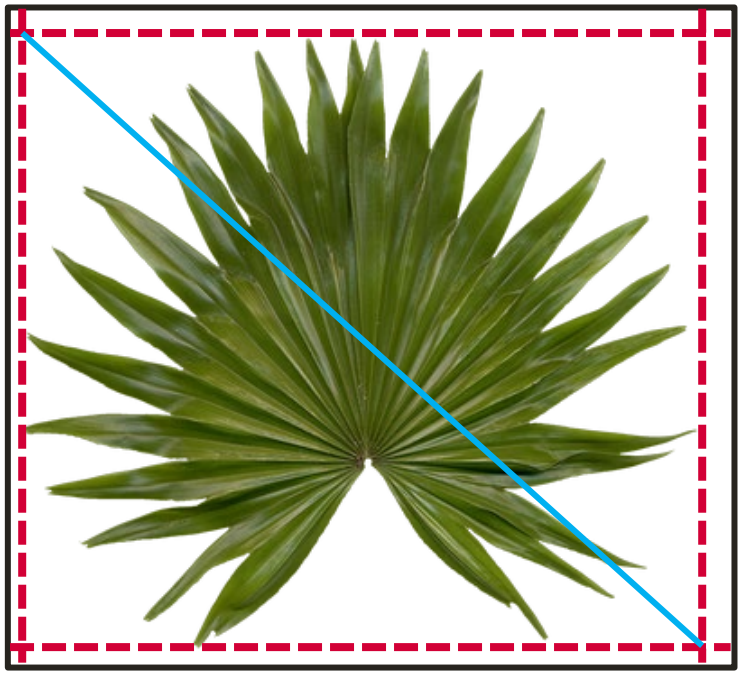
All were assumed to be mapped to pairs of triangles

- Trimmed off some of the 'nearly transparent' (8bit Alpha <= 30) borders.
- Near opaque (>=225) were clamped to fully opaque.
- Eliminates some spurious outliers

Using just pairs is a compromise but

- Don't have meshes for many of the textures
- Gruen et al pointed out that some models aren't authored that well anyway.
- Subdividing would appear to make things easier.

Post-processed data set - reduced to 3 levels – to be uploaded with paper.



Performance evaluation assumed equal distribution of samples across the squares.

Example Scene: San Miguel

There may be considerable depth complexity

Example views:

1. Alpha test off (black == transparent portions of textures)
2. Wire frame (with hidden surface removal)
3. Wire frame - to show scene foliage depth complexity

Can require a **significant** number of any-hit shader tests.

Final VQ4 Codebook Arrangement

Uniform O and T 4x4 vectors are again implicit

… and for the present, so is uniform C.

- **Uses indices {0,1,2}**

The explicitly stored vectors must meet the 247-bit budget.

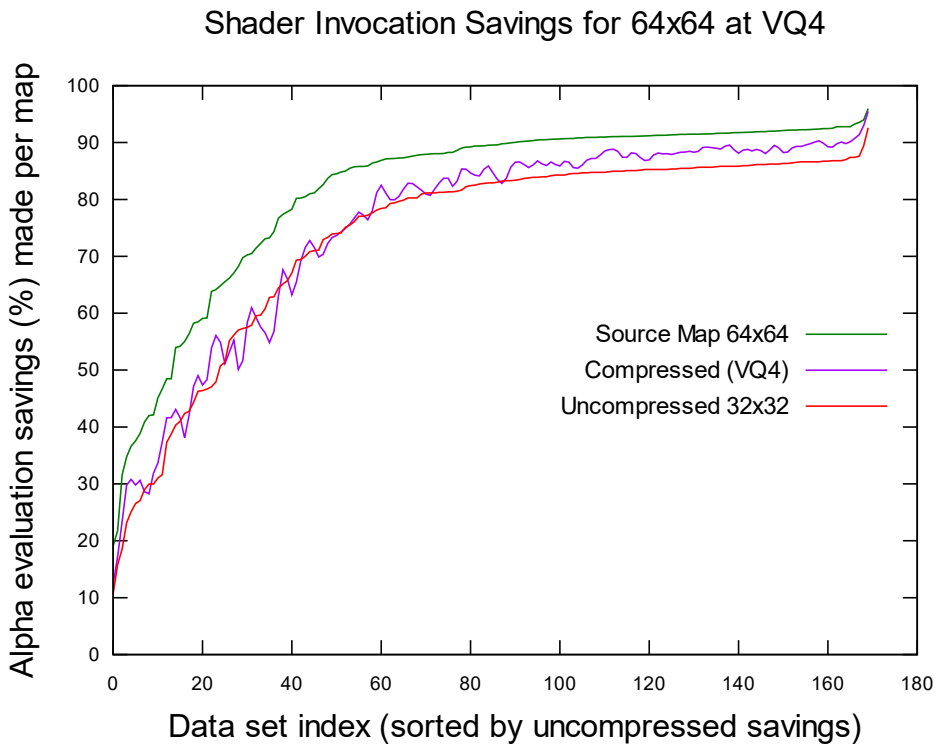
- **Combination of 20, 18 and 17-bit vectors**

Unlike VQ2, there are no-per-quadrant codebooks.

3x Implicit Uniform vectors **0** bits
3 ‘two value’ vectors 3x17bits
1 Vertical partition vectors 18 bits
1 Horizontal Partition 18 bits
8 General vectors 160 bits

Other resolutions

Note: 32x32 @ 2bpp is larger than 64x64 VQ4



Wavelet Experiment

Three level wavelet