NVIDIA

Micro-Mesh Asset Pipeline Toolkit

DISCLAIMER

Please refer to "Micro-Mesh Basics" slide-deck first

https://developer.download.nvidia.com/ProGraphics/nvpro-samples/slides/Micro-Mesh_Basics.pdf

Extended Content Pipeline

Opacity and Displacement micromaps operate in barycentric space following a spatial curve ("bird curve"), rather than uv/texture space. Maps are indexed directly and not sampled!

New barycentric data exchange file format (~= textures)

- Container for many triangles' data
- Per-micro-vertex (displacement) or per-micro-triangle (visibility) data
- Support uncompressed and hw-compressed formats

New properties in 3d model files

- Per-triangle data to map mesh to appropriate bary lookup
- Per-vertex displacement directions (fp16)
- Extend mesh with barycentric opacity / displacement

New support in tooling /micromesh sdk

- Generate/exchange above 3d model properties
- Generate/exchange barycentric files

File formats

Barycentric data containers:

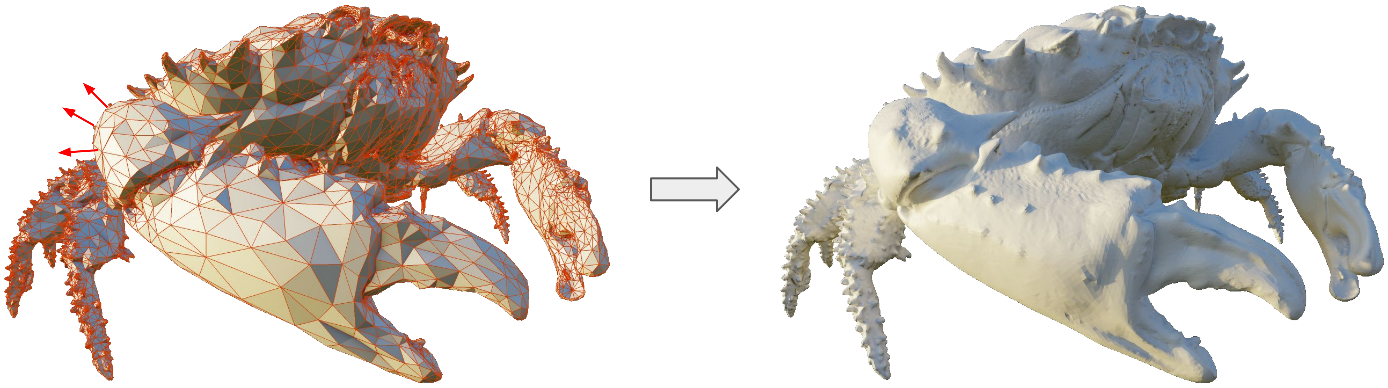
- BARY container
  - Refined for needs, target quick loading in APIs
  - Compressed and uncompressed data (per micro vertex / triangle)

3D scene formats

- GLTF extension for displacement directions & barycentric mapping
  - Focus on quick loading for APIs as well
  - Lightweight easy to integrate loader/savers for internal tests/prototyping
- USD schemas for editor / tool exchange
  - Self-contained MicromeshAPI schema embeds DMM properties
  - BarycentricFileAPI schema allows referencing .bary files

BARY

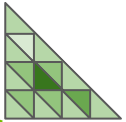
Micro-Mesh Asset

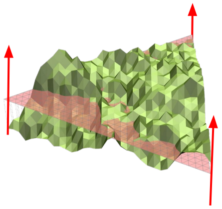


Model courtesy of threedscans.com

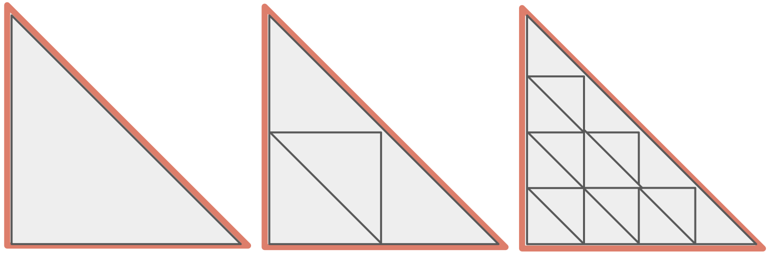
⧅ Base Mesh + Displacement Attributes + Displacement Micromap → ▩ Displaced Micro-Mesh

◻	Vertex Direction Vectors		Micromap::Triangles
⇆	Vertex Direction Bounds	△	Subdivision Level
b010	Triangle Primitive Flags	⇶	Displacement Values
[0,1,2]	Triangle Mapping Index		Compression Blockformat





Micro-Mesh Subdivision



Base triangle subdivision levels 0, 1 and 2

△ Subdivision Level	Triangles	Vertices
0	1	3
1	4	6
2	16	15
3	64	45
4	256	153
5	1024	561

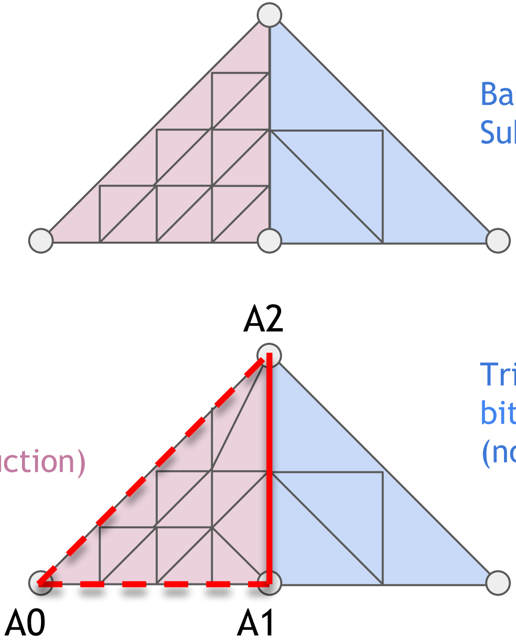
A micromesh is the result of subdividing a base triangle in power of twos along all edges uniformly.

Displacement supports up to level 5 in the raytracing APIs

Micro-Mesh Adaptive Subdivision

Base triangle A:
Subdivision level 2

Base triangle B:
Subdivision level 1

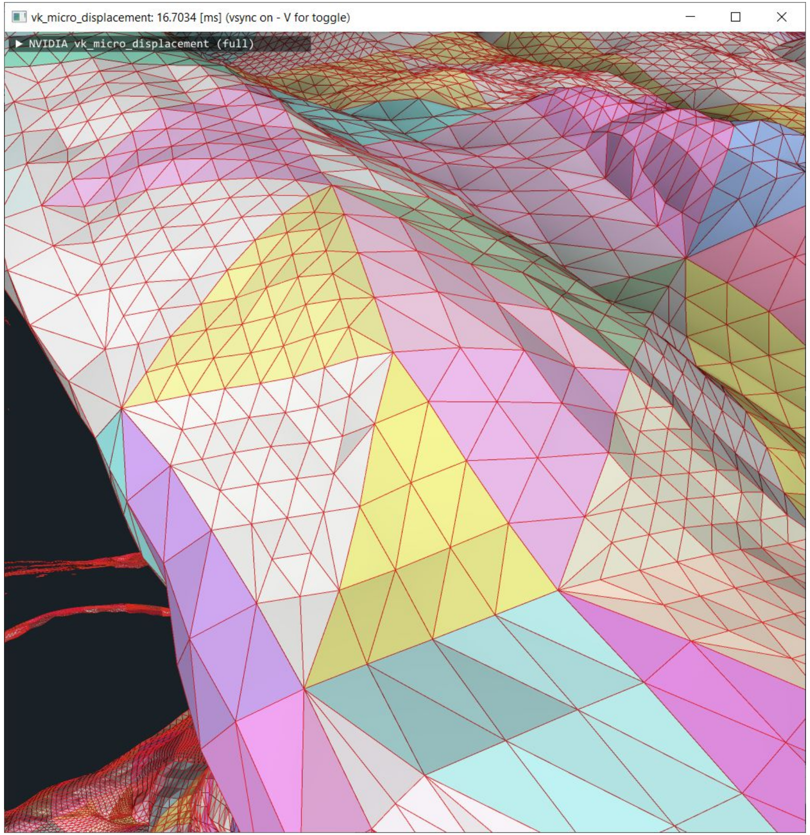


Triangle Primitive Flag:
bits 010 = 2
(second edge needs reduction)

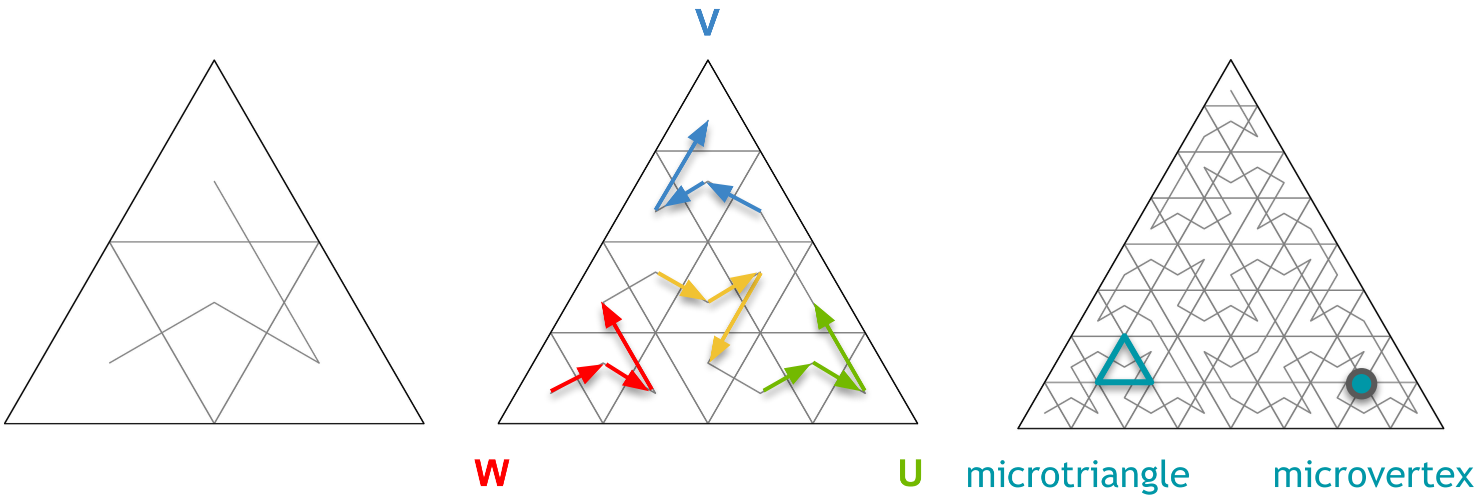
Triangle Primitive Flag:
bits 000 = 0
(no reduced neighbor)

Triangle Primitive Flag (u8) provides information for watertight intersections if one subdivision level difference exists between adjacent triangles.

Micro-Mesh Adaptive Subdivision



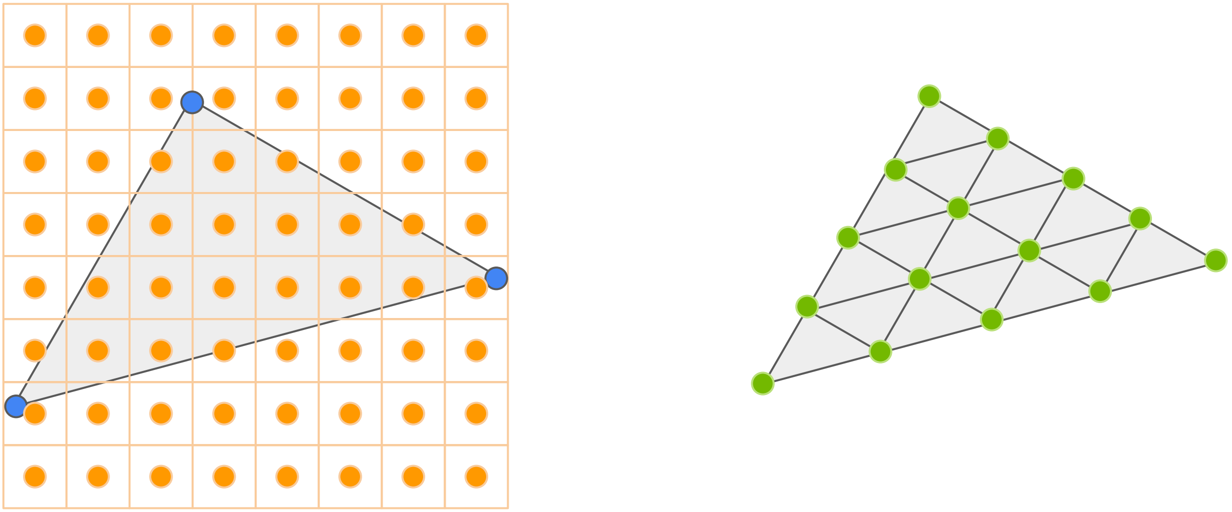
Micromap Values



The micromap values are ordered along a spatial curve (“bird-curve”) based on integer UV coordinates within triangle. The values are fetched/indexed and not sampled.

Displacement Micromap: per microvertex **scalar displacement values**
Opacity Micromap: per microtriangle opacity

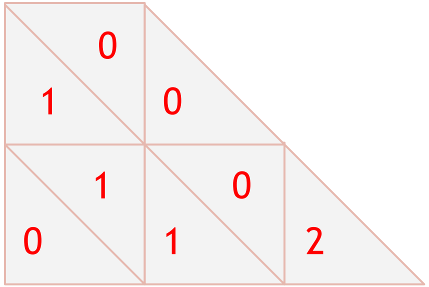
Micromap Values



Texture sampling is invariant to object-space orientation and triangle winding due to texture coords.

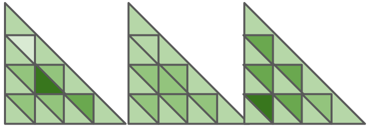
Micromap fetching is always in barycentric space.

Micromap Indexing

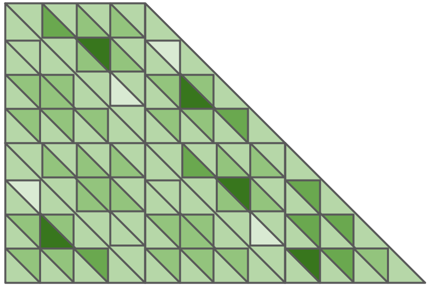


Base Triangles

`Mapping Index Buffer [0,1,1,0,2,1,0,0]`



Micromap Triangles 0, 1, 2



Micromap Applied to Mesh

Base triangles can use Triangle Mapping Index (u32) to reuse Micromap values.
It is a bit like UV-coordinates allowing to reuse sections of a texture.
Caveat: triangle vertex ordering matters!

Without mapping index a 1:1 mapping between base triangle and micromap triangle is effective.

Micromap Compression

Block Format	Microtriangles
64_MTRIS_512BIT (3-45 x uncompressed unorm11)	1-64
256_MTRIS_1024BIT (delta compression scheme)	256
1024_MTRIS_1024BIT (delta compression scheme)	1024

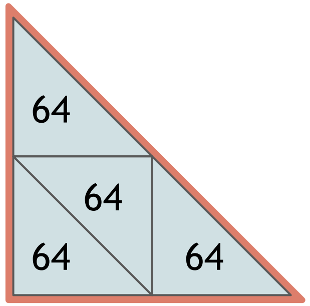
Example compression result:

base triangle (256 microtriangles)

→ split by compressor →

4 x 64_MTRIS_512BIT

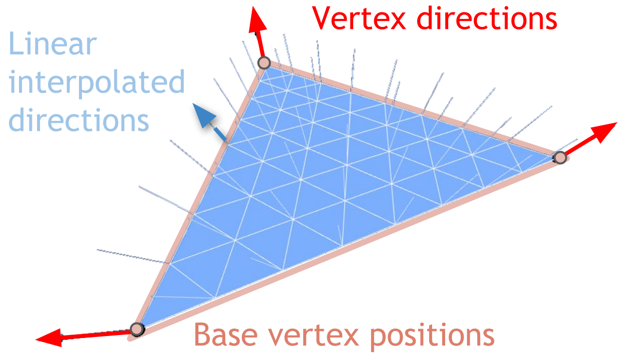
blocks follow “bird-curve”

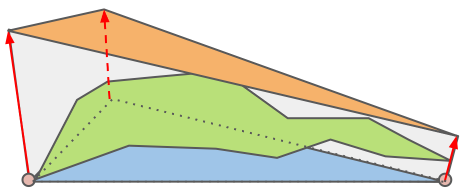


256 = 4 x 64

The displacement values are UNORM11 and block compressed.
The micromap **must be compressed for RT APs.**

Micro-Mesh Displacement

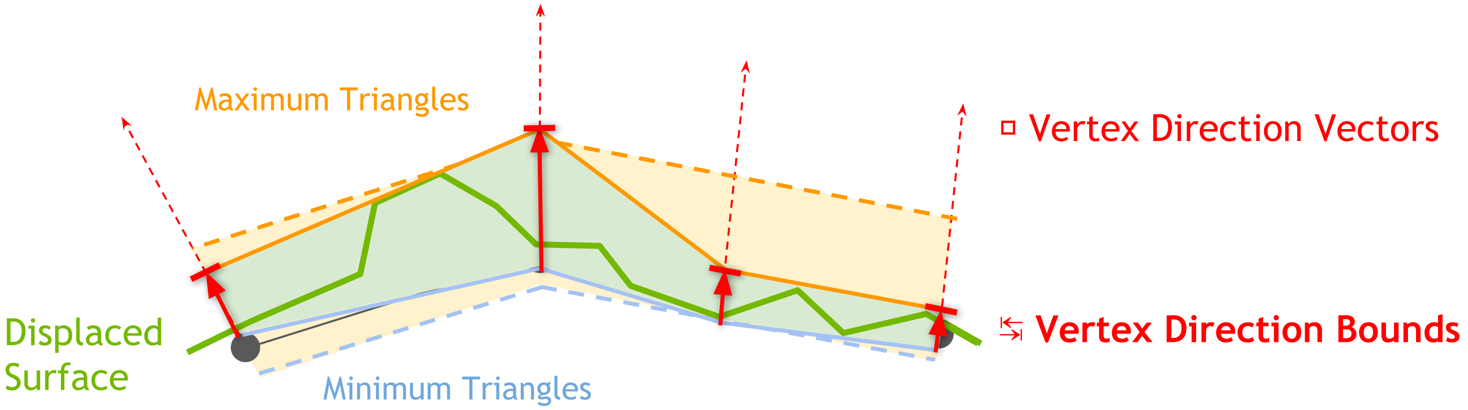




Vertex directions are linearly interpolated over base triangle and **their length matters**.

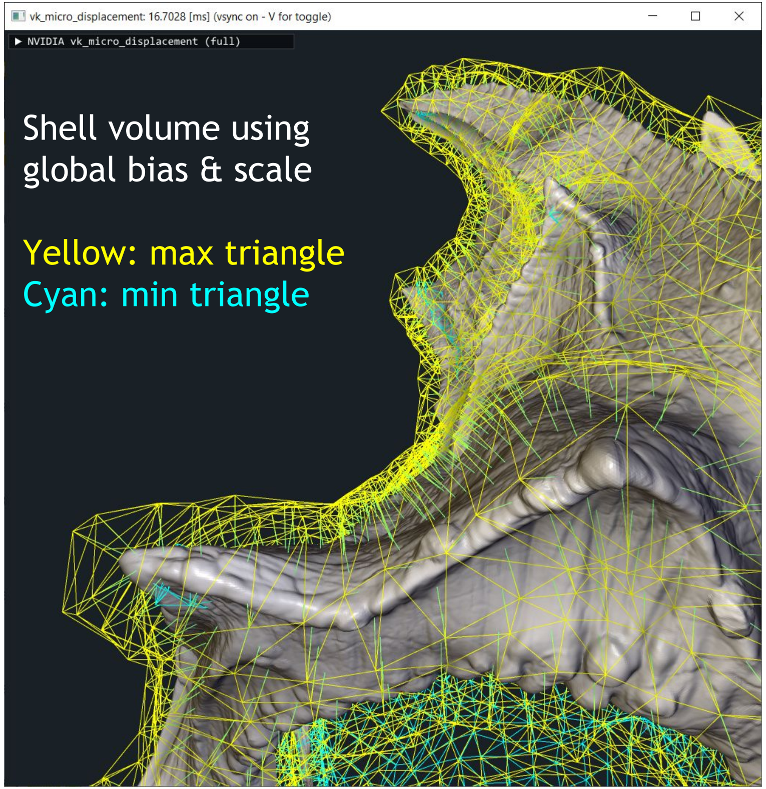
Each microvertex has a displacement value in range of [0,1] and therefore lies within a min and max triangle. The tighter this shell volume, the faster hw can raytrace (integer factors faster!).

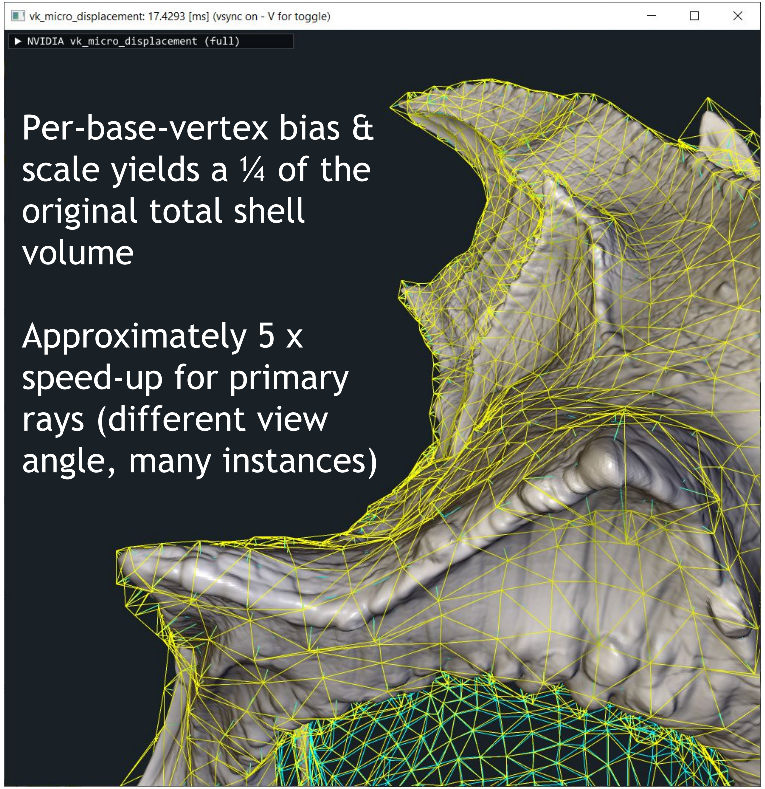
Micro-Mesh Displacement



This **cross section** shows importance of computing vertex direction bounds (bias & scale).
The shell volume is smaller with per-vertex bias & scale, than global bias & scale.

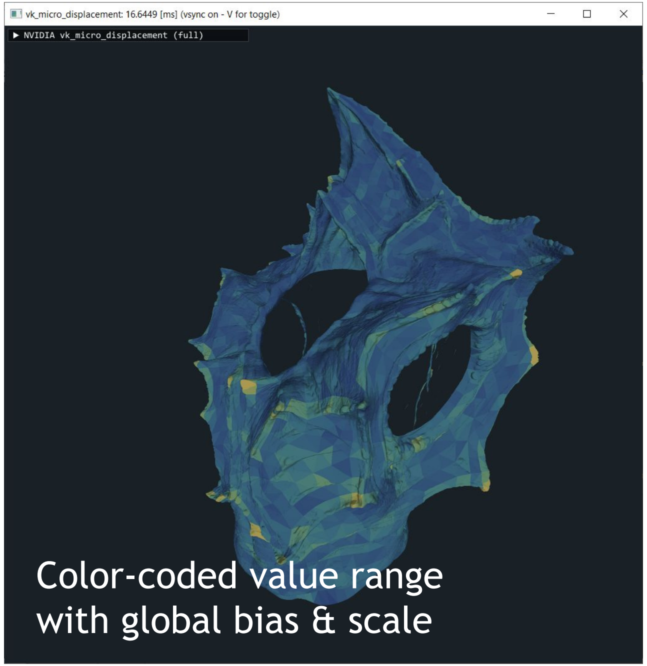
Direction Bounds Fitting

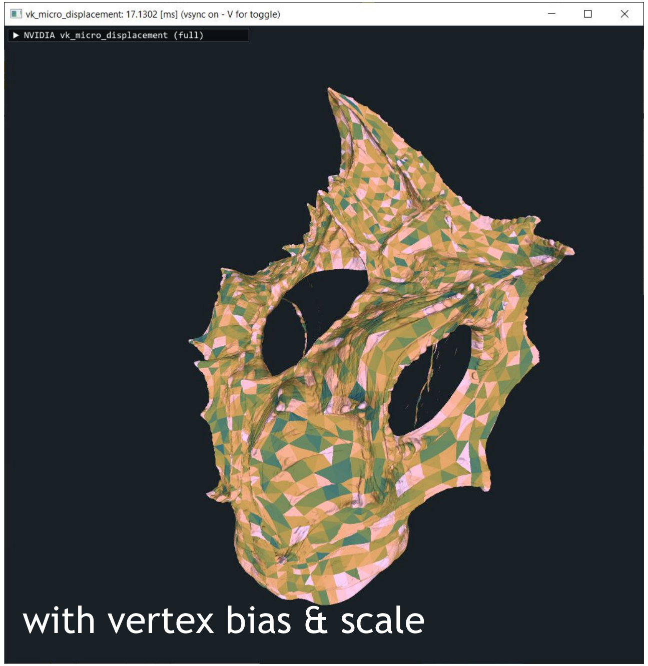




Model courtesy of Autodesk, armor part from “Turtle Barbarian” by Jesse Sandifer

Direction Bounds Fitting

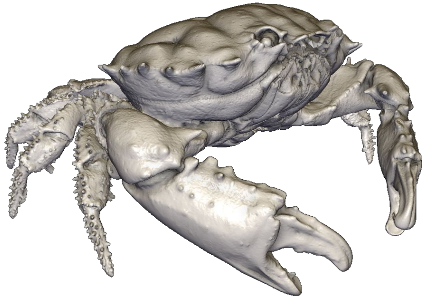


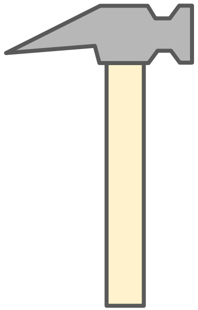


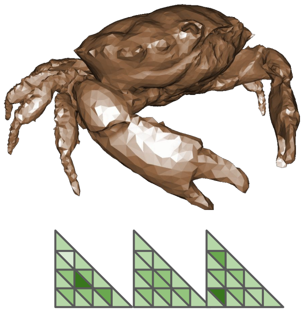
The fitting improves the effective value ranges, therefore precision used for UNORM11 as well

Micro-Mesh Asset Data Flow

Micro-Mesh Asset Data Flow

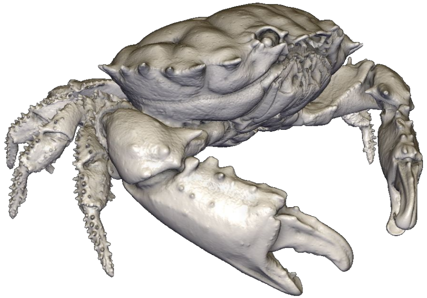






```
VkAcceleration
  StructureKHR
      +
VkAcceleration
  Structure
  Triangles
  Displacement
  MicromapNV
```

```
VkMicromapEXT
```



Reference Mesh
Typically hi-resolution, or mid-resolution with heightmap displacement ("substances")

Displacement Toolkit
Generates all required data for rendering.

Can operate beyond RT-API feature set (higher subdiv levels, float data, microvertex normals…)

Baked Mesh
Base Mesh,
Displacement Attributes,
Displacement Micromap

Runtime Mesh
Raytracing API objects

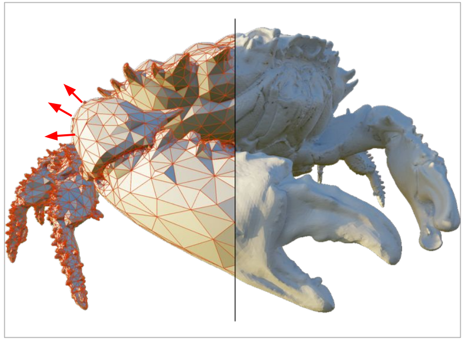
Load baked data directly into RT APIs.

Rendered Mesh
Ada:
hardware interprets data directly

previous gen:
driver-side intersection shader

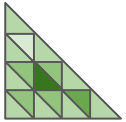
Micro-Mesh Processing Pipeline

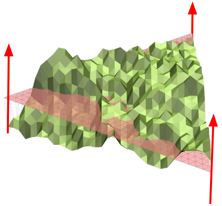
Micro-Mesh Asset



⧄ Base Mesh + Displacement Attributes + Displacement Micromap → Displaced Micro-Mesh

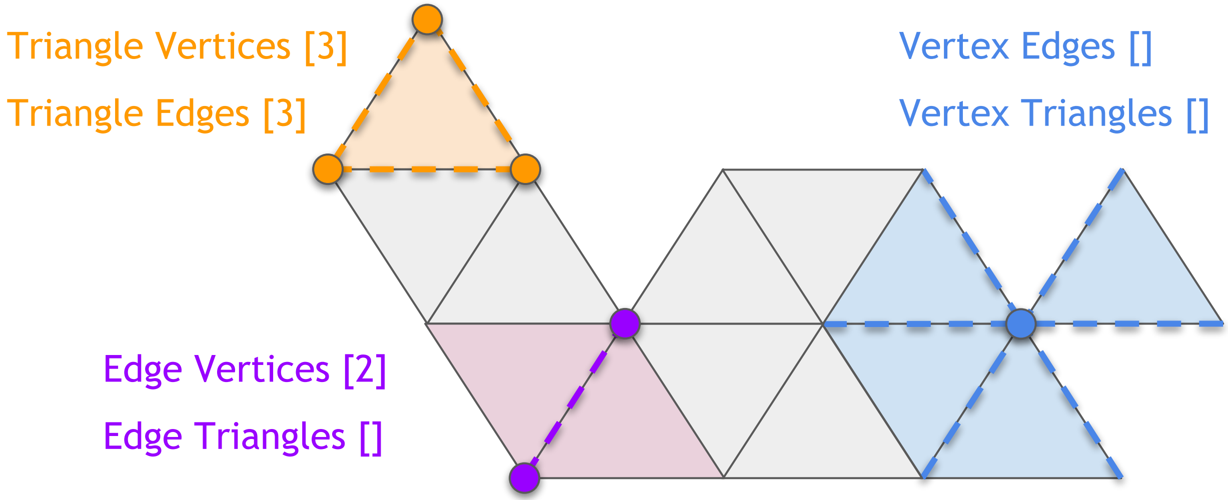
◻	Vertex Direction Vectors	Micromap::Triangle
⇤⇥	Vertex Direction Bounds	△ Subdivision Level
b010	Triangle Primitive Flags	⇶ Displacement Values
[0,1,2]	Triangle Mapping Index	Compression Blockformat





Currently not fully supported by all operations in toolkit → always uses 1:1 mapping

Mesh Topology



The MeshTopology data structure provides connectivity information within a mesh.

Several operations require it to ensure the overall mesh adheres to certain consistency rules.

Allows non-manifold inputs. SDK can build it, or its provided by user.

1. Setup Input

▩ Reference Mesh
(optionally displaced by heightmap texture)

2. Pre-Tessellation

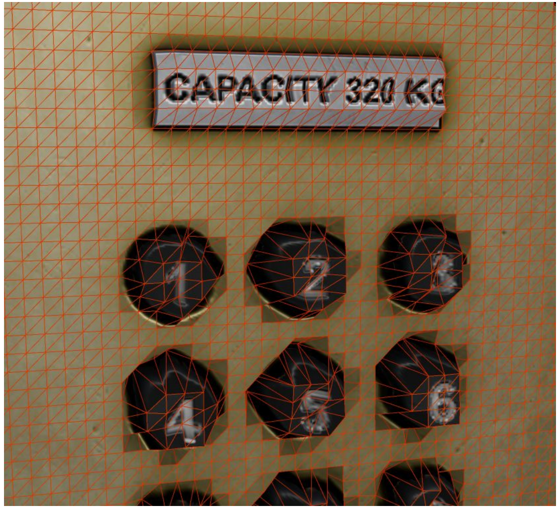
Add triangles due to subidv 5 limit.

⧄ Base Mesh

◘ Vertex Direction Vectors

◬ Subdivision Level Hints

Subdivision Levels & Heightmaps



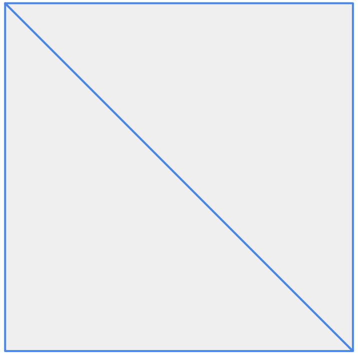
Model courtesy of Adobe
and Jonathan BENAINOUS

Increasing subdivision level lowers aliasing artefacts.

But we are limited to level 5 (1024 per input triangle), equivalent to 32 x 32.

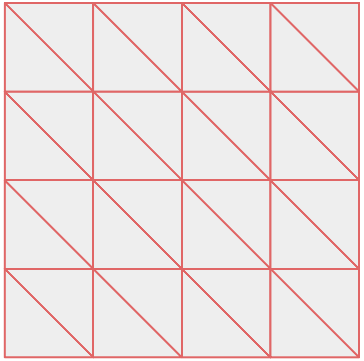
A simple quad with a 4096 x 4096 displacement map needs at least level 12

Pre-Tessellation



Reference mesh has
▦ Heightmap resolution
of 128 x 128

△ Subdiv Level 7 per
triangle



Base mesh is
pre-tessellated

△ Subdiv Level 5 per
triangle

Pre-Tessellation tessellates the reference mesh where the reference triangle would be under served with subdiv level 5.

Note: changing the mesh is invasive, what about vertex skinning information, morph targets...

Pre-Tessellation

Only illustrative white wireframe of original reference mesh

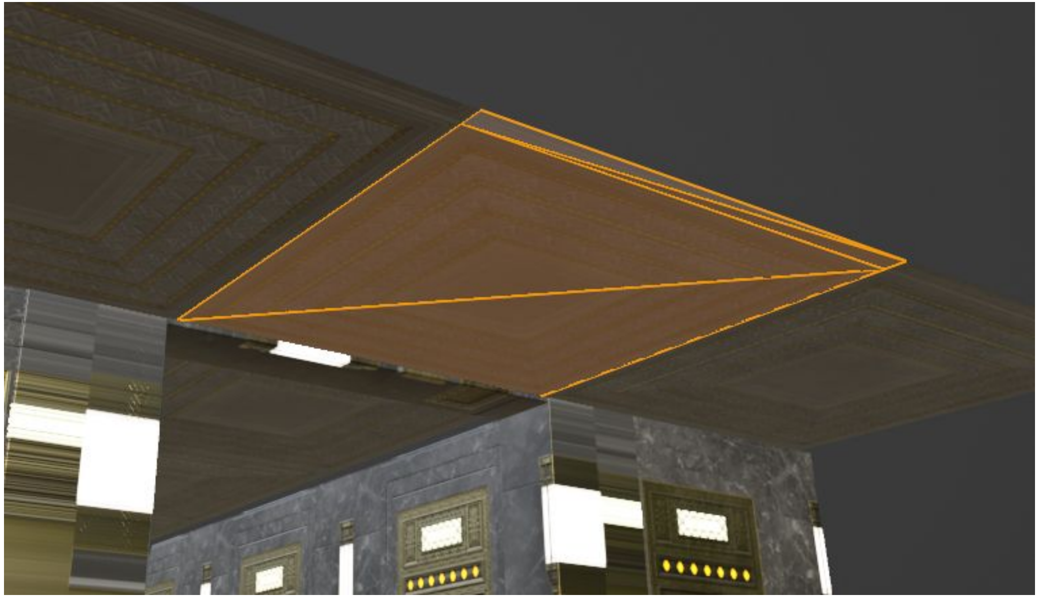
Model courtesy of Adobe and Jonathan BENAINOUS

Pre-Tessellation Caveats

What about non-square areas?

Uniform tessellation would yield long skinny triangles for the upper trim quad.

→ Needs quad detection (we experimented with this)

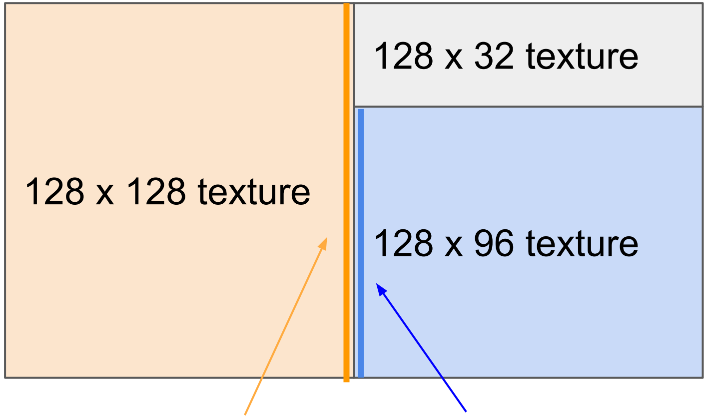


Model courtesy of Adobe and Jonathan BENAINOUS

Pre-Tessallation Caveats

What if UV edge needs non power of 2?
What if adjacent quad is a different mesh…
What if the DCC tool shows no gap, but micro-mesh does?

→ We stuck to simple triangle subdivision

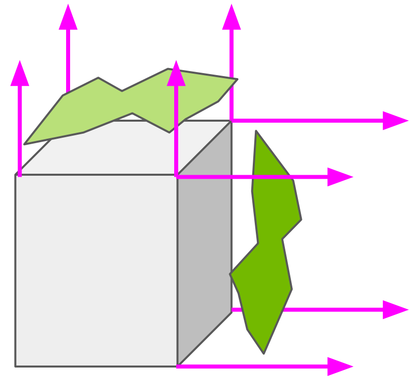


With uniform power of 2 subdivision both edges end up 128

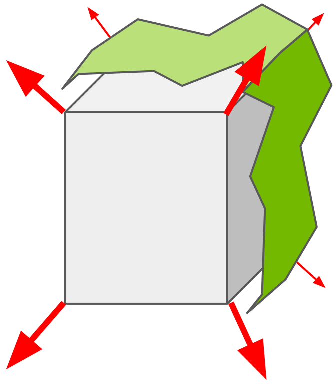
Note: non-power of 2 texture may not be that common, but artist can build an atlas and use UV coordinates however they see fit

Vertex Directions

Many applications use vertex normals, however these can cause cracking artefacts along edges. Some applications close these afterwards through strips, or other means.

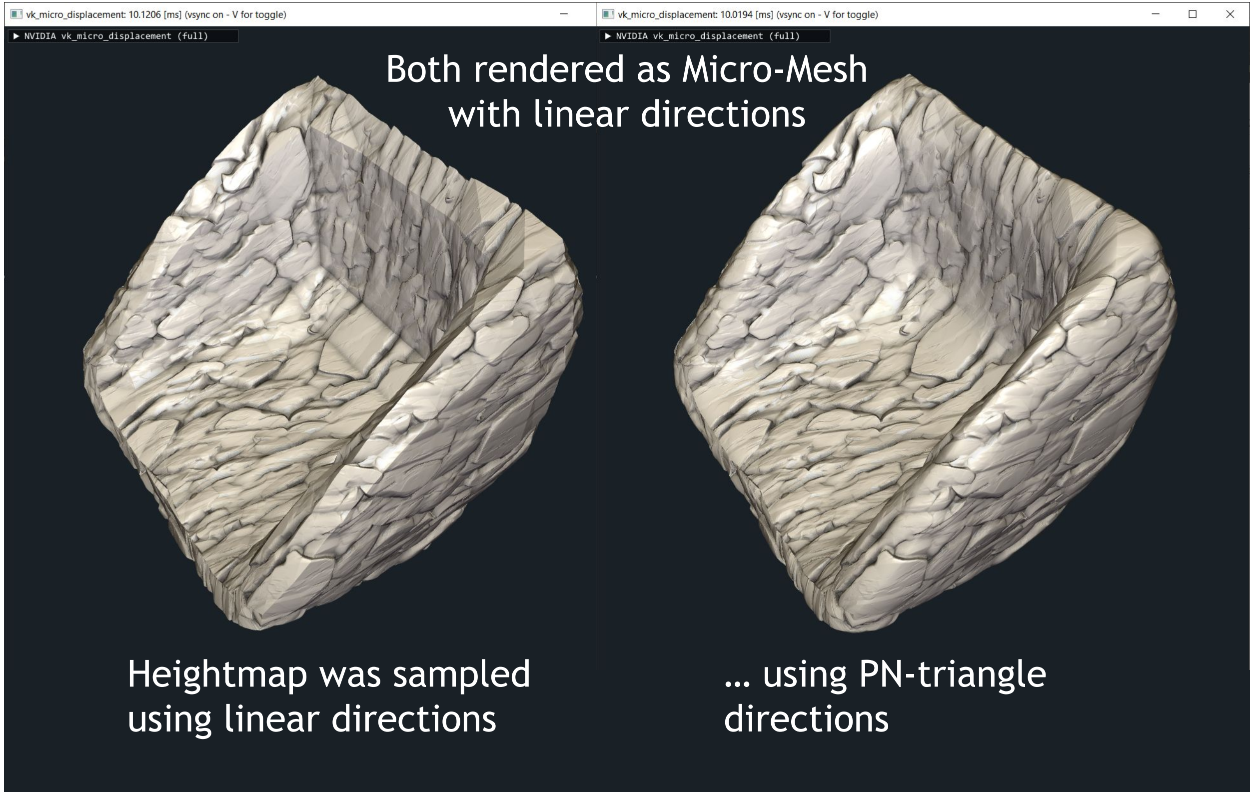


We use smooth normals for vertex directions as they ensure watertightness is preserved.



Vertex Directions & Heightmaps

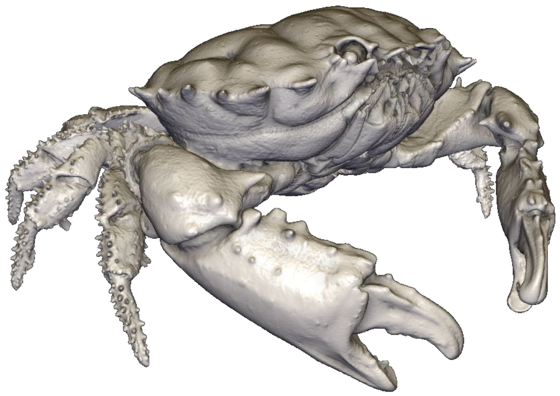
When heightmaps are used, we can use more advanced interpolation, including normalization, since it will be resampled into linear Micro-Mesh space.



Substance material courtesy of Adobe

1.	Setup Input		▩ Reference Mesh (optionally displaced by heightmap texture)
2.	Pre-Tessellation	Add triangles due to subidv 5 limit.	⧅ Base Mesh
	Remeshing	Reduce number of triangles.	◘ Vertex Direction Vectors ◬ Subdivision Level Hints

Remeshing



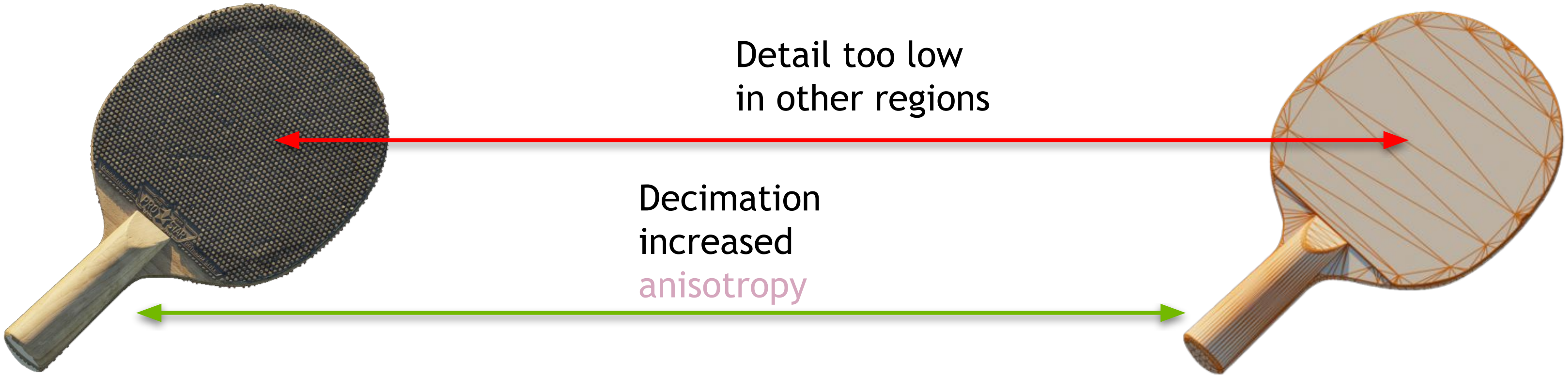
Reference mesh with millions of triangles

Remeshed base mesh with typically 0.1 % to 1 % of triangles.

Micro-Mesh is mostly about compressing high-detail.

Remeshing turns the high resolution reference mesh into a lower resolution base mesh.

Remeshing (Early Test)



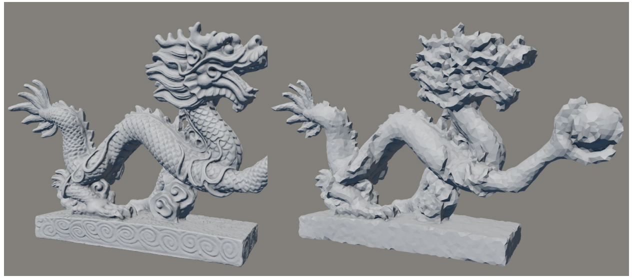
Hi-res mesh with heightmap

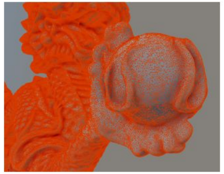
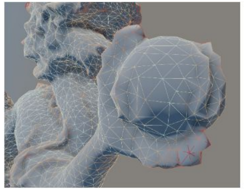
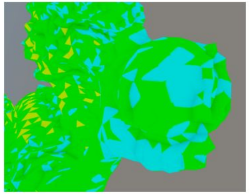
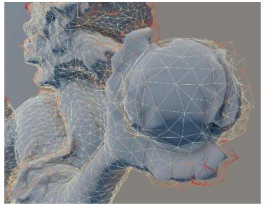
Traditionally decimated

Standard mesh simplifiers tend to introduce anisotropic (long/skinny) triangles, as texture detail can still be projected well. Not good for Micro-Mesh. Remeshing provides lots of challenges.

Remeshing

GPU-based remesher to overcome the performance bottleneck with high complexity models. This became our toolkit solution. Warning: currently not deterministic, results will vary

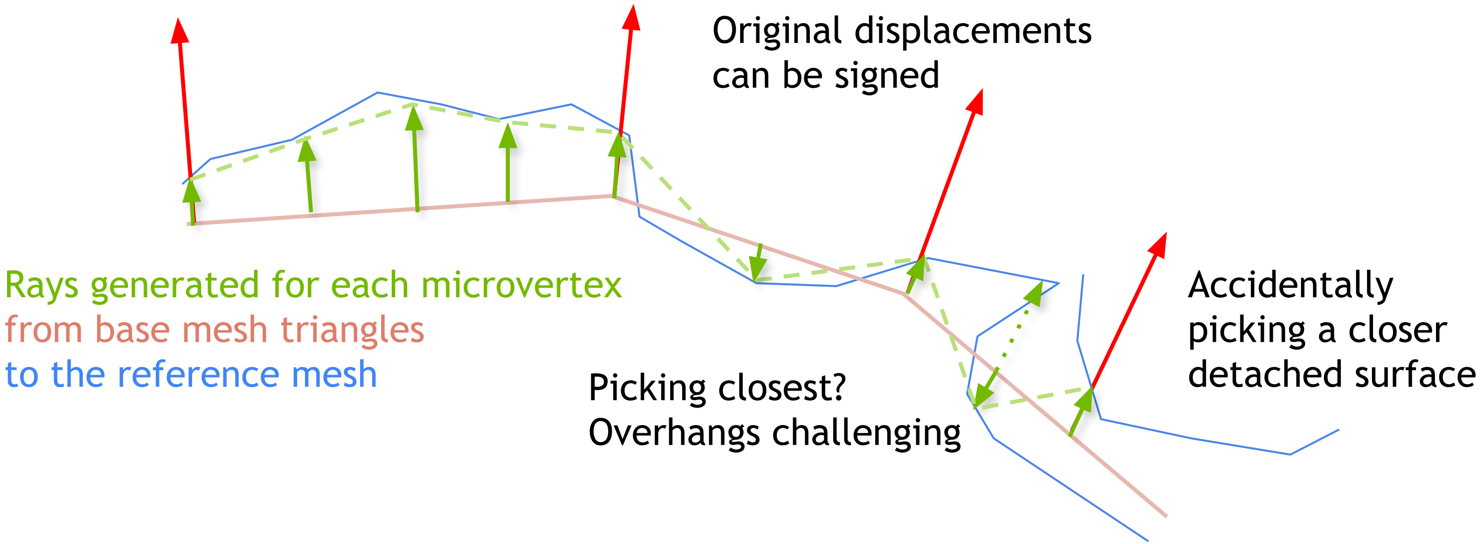


Original Mesh	Decimated Result	Target Subdivision Level	Displacement Bounds
			

GPU remesher: 2.4 M to 20 K triangles in 160 ms, with micro-mesh metadata

1.	Setup Input		▧ Reference Mesh (optionally displaced by heightmap texture)	
2.	Pre-Tessellation	Add triangles due to subidv 5 limit.	⧄ Base Mesh	
	Remeshing	Reduce number of triangles.	◻ Vertex Direction Vectors	◬ Subdivision Level Hints
3.	Subdiv - Sanitization	Enforce +/- 1 subdiv level difference between triangles.	b010 Triangle Primitive Flags	△ Subdivision Level
4.	Micromap Baking w. Bounds Fitting (w. Texture Resampling)	Raytrace from base to reference. Adjust directions / fit bounds to displacement values.	⇆ Vertex Direction Bounds	Raw Displacements

Baking



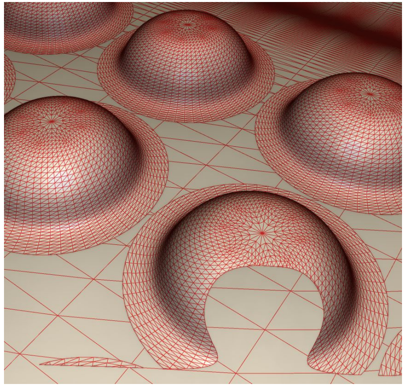
The baker uses raytracing to generate the micromap displacement values.

Baking causes a lot of tricky situations, commercial tools expose several means for artist to get desired result.

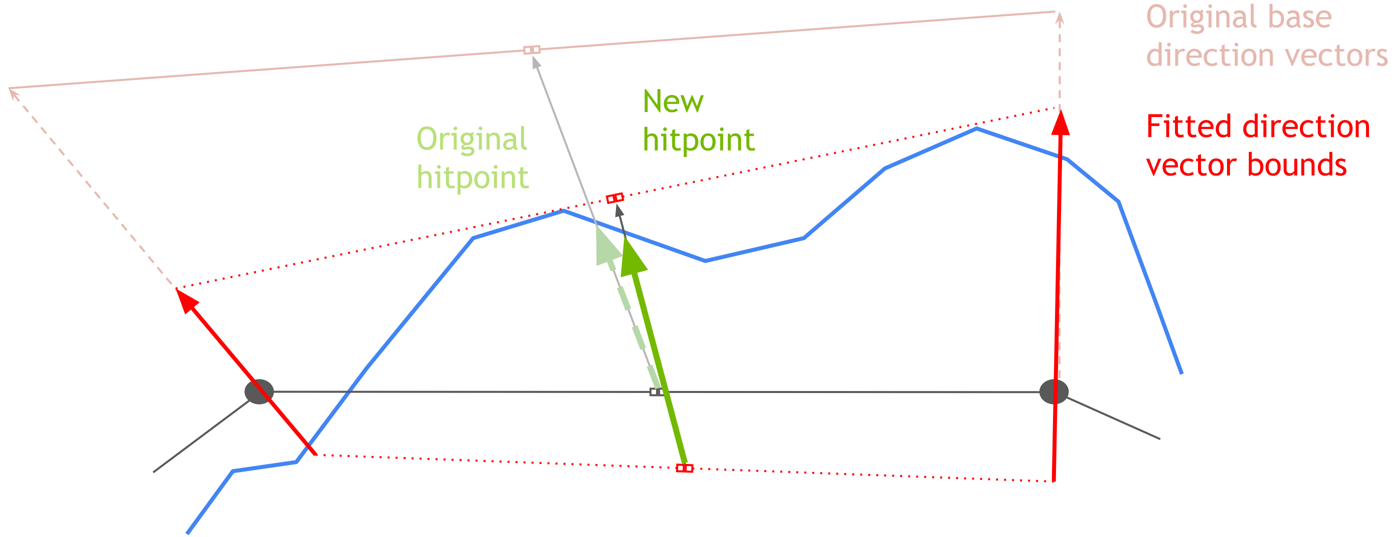
Remesher can help by providing bounds from reference mesh within the rays can stay. But still just ray ranges.

Baking

Some artefacts are intentional, e.g. “floater” geometry to add detail cheaply, as it works well for normal maps.



Direction Bounds Fitting

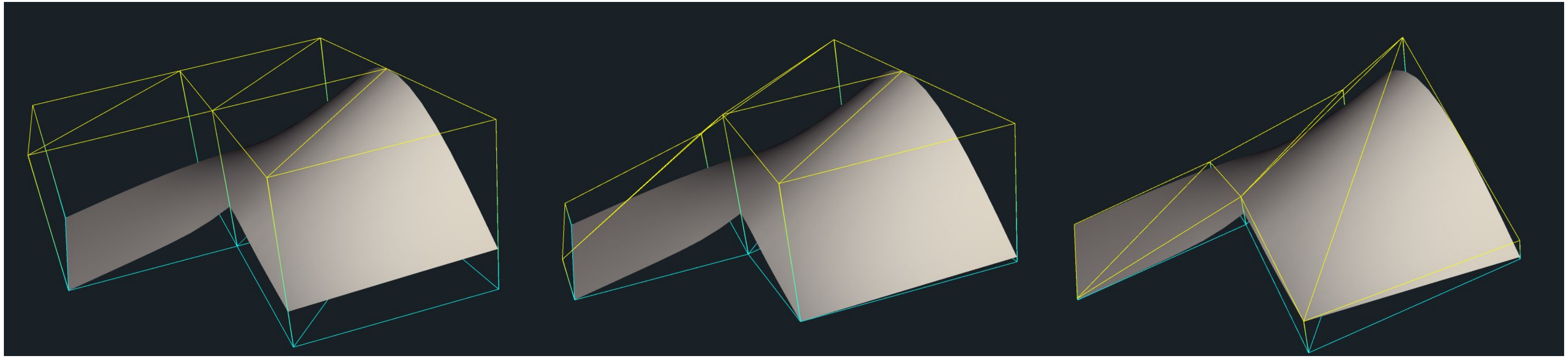


During baking process retrieve per-triangle min/max and atomically distribute to per-vertex min/max.

Use this to create vertex direction bounds (`bias = min` & `scale = max-min`).

Might need to iterate as bounds do change ray directions (unless all parallel, i.e. planar heightmap)

Direction Bounds Fitting



Global bias & scale

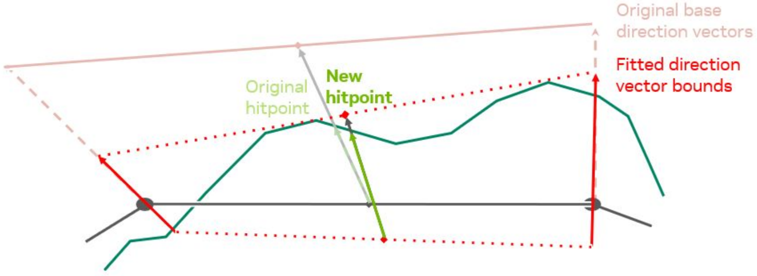
Simple triangle min/max to per-vertex min/max

(was most robust on real-world data)

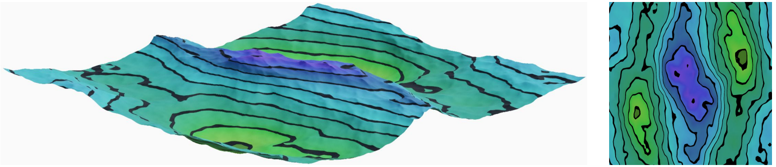
Plane upper and lower fitting per triangle to per-vertex min/max

(a bit brittle on real-world data, often worse)

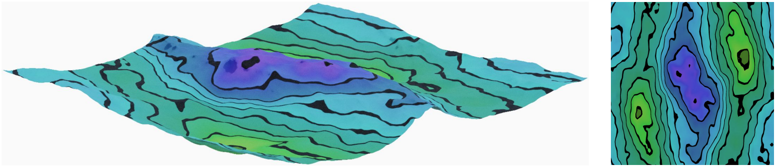
Texture Resampling



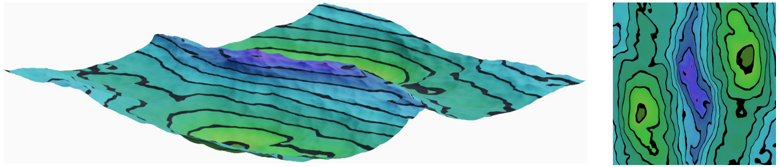
Remeshing and bounds fitting makes original texture coordinates less usable. As a result old textures will show artefacts. Need to resample textures or create an indirection / UV offset texture.



Reference (hi-res)

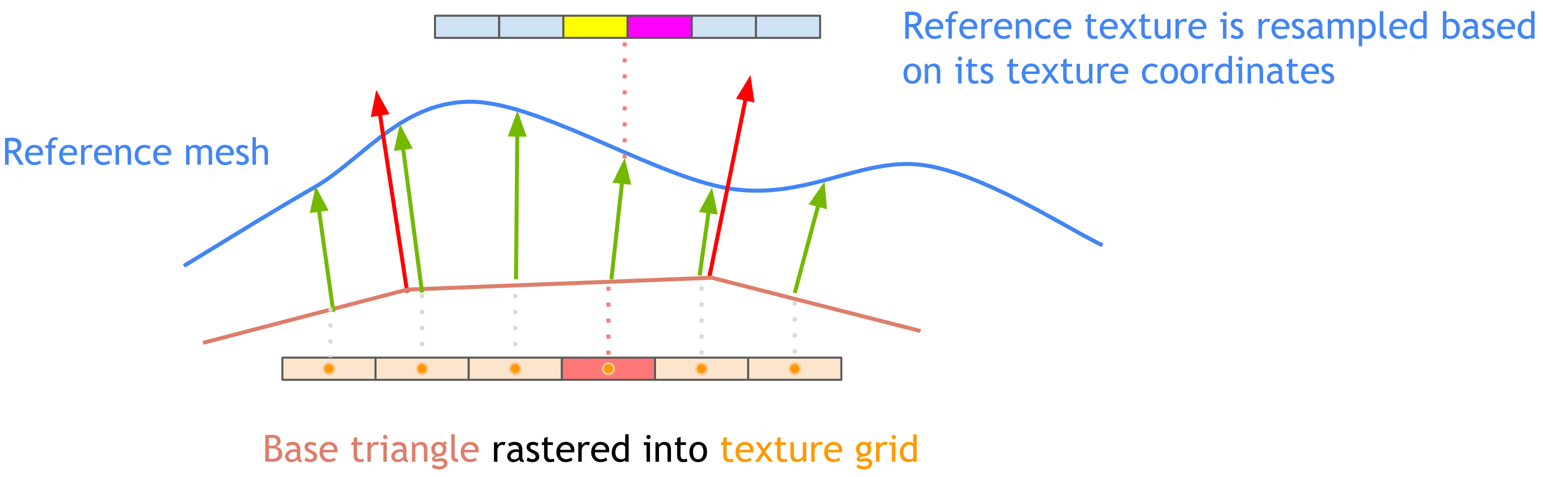


Baked without resampling



Baked with resampling

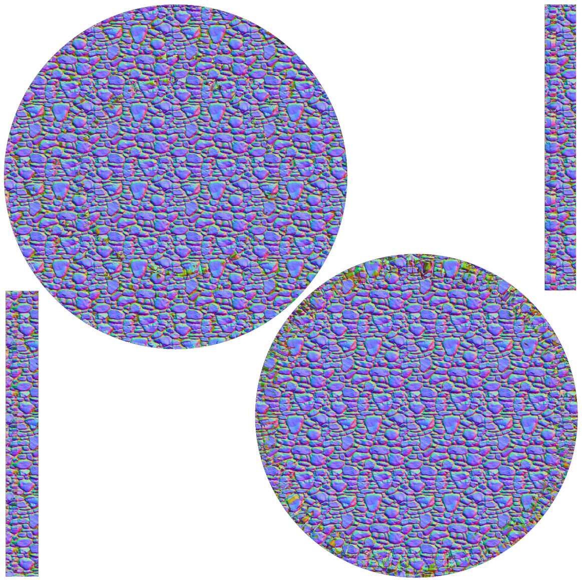
Texture Resampling

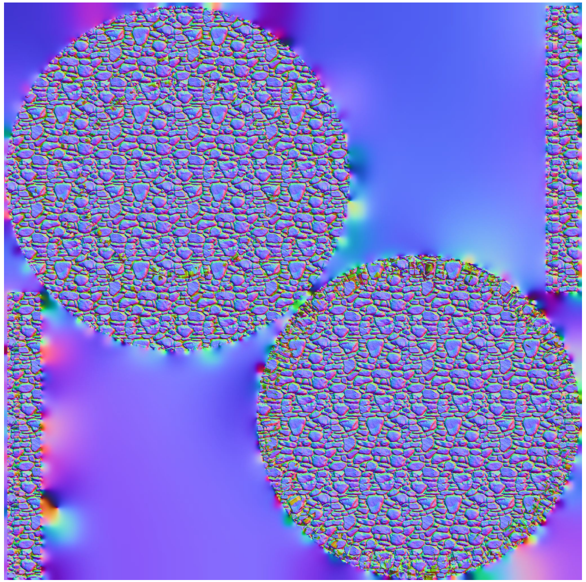


Resampling rasterizes texture coordinates of base mesh as positions so that each pixel represents a texel of the destination texture. Uses raytracing like baking.

For normal maps, also need to account for tangent space in both reference and base mesh.

Texture Resampling





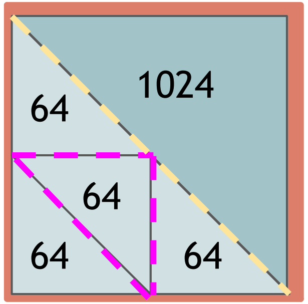
New textures are processed through pull-push filter, so that they create better mip-maps and no sampling artefacts along texture coordinates seams / between UV islands.

1.	Setup Input		Reference Mesh (optionally displaced by heightmap texture)	
2.	Pre-Tessellation	Add triangles due to subidv 5 limit.	Base Mesh	
	Remeshing	Reduce number of triangles.	Vertex Direction Vectors	Subdivision Level Hints
3.	Subdiv - Sanitization	Enforce +/- 1 subdiv level difference between triangles.	b010 Triangle Primitive Flags	△ Subdivision Level
4.	Micromap Baking w. Bounds Fitting (w. Texture Resampling)	Raytrace from base to reference. Adjust directions / fit bounds to displacement values.	Vertex Direction Bounds	Raw Displacements
5.	Value - Sanitization	Ensure values along edges between triangles match bit exactly.		Displacement Values
6.	Optimization & Compression	Block compress value based on user's quality settings.		Compression Blockformat

Compression

1024 = 1 x 1024_MTRIS_1024BIT (lossy)

Edges between inner blocks



Edges between outer blocks

256 = 4 x 64_MTRIS_512BIT (lossless)

Model courtesy of the Smithsonian Institution Digitization Program Office

Compression library must ensure that values along edges between triangles match as well as edges between the block triangles. Consumes UNORM11 displacements and uses direction magnitude to guide quality metric to pick one block format per micromap triangle.

Finalization

Some extra data should be computed at the end for the final baked asset:

→ A blockformat & subdiv level histogram which the raytracing APIs require.

→ Triangle min/max displacement values are useful for culling & LoD in rasterization.

→ Uncompressed displacement values for level 0 to 2 speed up sw-decoding in rasterization shaders.

Content

Model courtesy of the Smithsonian Institution Digitization Program Office

Photogrammetry Scans

→ Less aliasing artefacts due to typically organic / noisy surfaces
→ Hi-res texture inputs tolerant to resampling
→ Challenge is cleanup to have well behaved meshes (non-manifolds, self overlapping etc.)

Traditional Heightmaps

→ Can work well with pre-tessellation / flat surfaces
→ Problematic can be modularized assets, combining quads with different heightmaps etc.
→ In general displacement / height maps not as well standardized in behavior (cracks between seam edges, interpolation behavior, directions etc.)

Substance material courtesy of Adobe

Content

Hardsurface Modelling / CAD

→ Triangle orientation influences sampling grid, prone to aliasing for artificial / machined objects
→ Scalar displacement insufficient for sharp edges
→ CAD typically has shading normals not provided as normalmap textures, can be significant amount of microvertex data / requiring shader-based compression scheme at render time.

Game Content

→ Micro-Mesh displacement is not an “added detail effect”, the content is changed significantly to make it work. Having two separate sets of assets is a no go.
→ Too invasive as afterthought, developer must opt-in early to just have micromesh assets

Micro-Mesh SDK & Toolkit

Micro-Mesh SDK Components (some TBD)

micromesh_core

micromesh_displacement_remeshing

micromesh_displacement_compression

Provides most functionality to generate or process micromaps (barycentric data) with 3d meshes.

C-ish APIs for easy integration and multi-threaded.
Pointer & stride interfaces to avoid memory allocation.
All inputs and outputs strictly allocated outside API.
CPU & GPU processing with API agnostic interface.

bary_core

gltf 2.0 extensions

USD schema

Basic file container to exchange micromaps.
Extensions to 3D file formats.

micromesh_tool

micromesh_toolbox

Python binding

(samples)

UI and console apps.
Serves as sample code for tool developers.
Showcases rendering integrations.

Toolkit Design

The current toolkit is designed to use a layered approach

Toolkit applications

Python/numpy binding

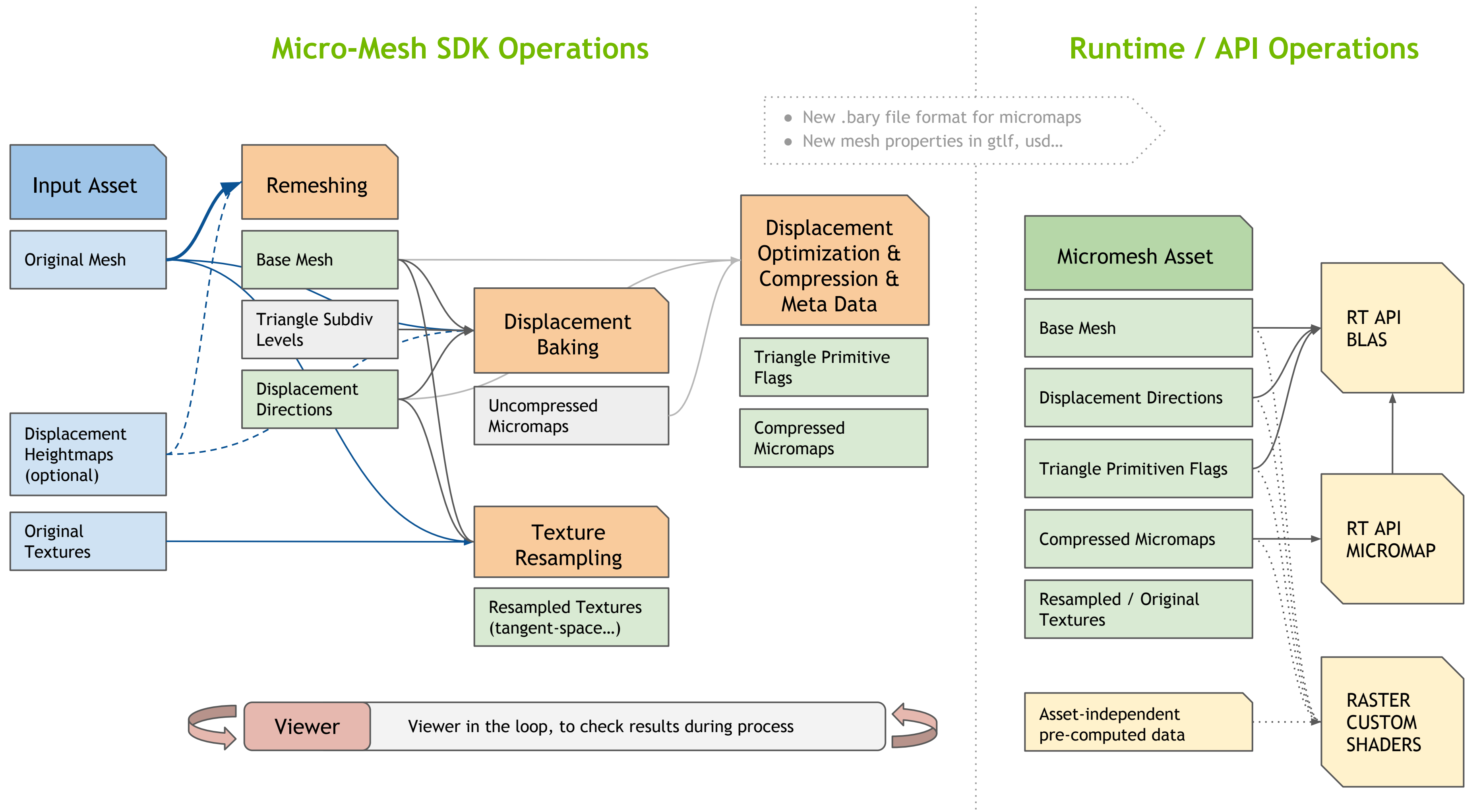
meshops libraries

bary libraries

micromesh libraries

High-level libraries to process meshes. Implemented using Vulkan and the low-level SDK libraries. Reference pipeline, internal use

Libraries designed to be embedded in third party tools


Micro-Mesh SDK Operations
Runtime / API Operations
New .bary file format for micromaps
New mesh properties in gtlf, usd...
Input Asset
Original Mesh
Displacement Heightmaps (optional)
Original Textures
Remeshing
Base Mesh
Triangle Subdiv Levels
Displacement Directions
Displacement Baking
Uncompressed Micromaps
Texture Resampling
Resampled Textures (tangent-space...)
Displacement Optimization & Compression & Meta Data
Triangle Primitive Flags
Compressed Micromaps
Viewer
Viewer in the loop, to check results during process
Micromesh Asset
Base Mesh
Displacement Directions
Triangle Primitiven Flags
Compressed Micromaps
Resampled / Original Textures
Asset-independent pre-computed data
RT API BLAS
RT API MICROMAP
RASTER CUSTOM SHADERS

Micro-Mesh SDK Components

micromesh_core

- Core API to work with Micromap data
- Provides most data types also used by other micromesh component libraries
- Utility classes and functions to sanitize values for watertight behavior or perform other typical operations for micromaps and micromeshes (e.g. tessellation, type conversions etc.)
- `micromesh::OpContext` class wraps managing some basic multi-threading for bigger operations.

- C-ish API design
- All input & output allocation done by developer
- Storage agnostic, uses pointer & stride interface (`micromesh::ArrayInfo`)
- Does **not** define / contain file format related code
- Does not depend on any file formats either

Micro-Mesh SDK Components

micromesh_displacement_compression

- Takes 11 bit unorm displacement inputs and compresses them into barycentric displacement blocks of 512 or 1024 bits.
- Uses provided mesh topology information to preserve a watertight representation
- Exposes a few control parameters to control quality

micromesh_remeshing

- Remeshing / mesh simplification algorithm that targets micromesh representations (favors isometric triangles)

Auxiliar Components

bary_core / bary_utils libraries

- bary_core, C-ish, pointer-based api, defines core of BARY file format definitions and io agnostic helpers
- bary_utils contains utilities using stl to load/save and work with data. Use as starting point for integrations
- Apache 2.0 License

Toolkit Components

meshops_core

- Utility library that provides an easier to use, higher level C++ interface to the micromesh sdk
- Central data structures are the `meshops::MeshView` related classes
- Provides CPU implementation for many basic operations

meshops_baker

- Features GPU-based micromap baking of displacement data and texture resampling

meshops_remesher

- GPU-based remeshing

meshops_optimizer

- TBD, optimize and compress displacement micromaps (currently implemented within micromesh_tool)

Micromap Data Structure

Micromap & BARY

The micromesh SDK uses pointer & stride interfaces to access data stored elsewhere

`micromesh::Micromap` can be extracted from a bary file

The following illustrations can therefore be easily applied to the format’s data structures.

These illustrations are similar to the ones from the basic slide-deck, they make use of “bary” namespace types where applicable. They do not introduce new concepts.

Micromap & BARY

Pseudo code for illustrative purposes

Micromaps

```
bary::ContentView {
  bary::Triangle triangles[];
  ValueType      values[];

  ... // some more meta info
};
```

```
bary::Triangle {
  U32 valuesOffset;     // starting location of values for this triangle
  U16 subdivisonLevel; // resolution of this triangle
  U16 blockFormat;     // only relevant for compressed values
};
```

Mesh Triangle Mapping

```
                                          // optional mapping buffer allows re-use of map data for different mesh triangles
bary::Triangle micromapTriangle = mappings ? micromap.triangles[ mappings [ meshTriangleIndex ] ]
                                                       : meshTriangleIndex;
```

Value Fetching (uncompressed)

```
ValueType* triangleValues = micromap.values[ micromapTriangle.valuesOffset ];
// index into triangleValues using the spatial storage layout based on microVertex coordinates
ValueType microVertexValue = triangleValues[ computeStorageLayoutIndex( microVertex.barycentricIntegerUV )  ];
```

Micromap & BARY

Pseudo code for illustrative purposes

Micromaps

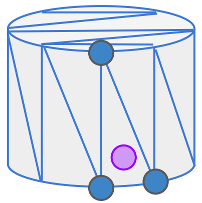
```
bary::ContentView {
  bary::Triangle triangles[];
  ValueType      values[];

  ... // some more meta info
};
```

```
bary::Triangle {
  U32 valuesOffset;    // starting location of values for this triangle
  U16 subdivisonLevel; // resolution of this triangle
  U16 blockFormat;     // only relevant for compressed values
};
```

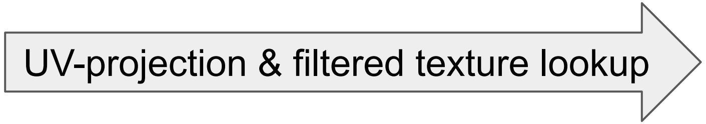
Value Fetching (compressed)

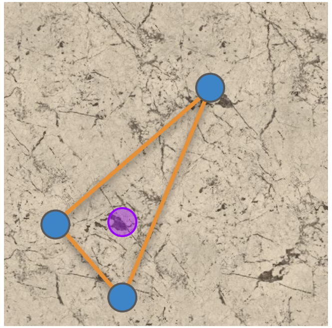
```
// compressed or specially packed values operate with byte offsets
Bytes* triangleValues = &micromap.valuesCompressedBytes[ micromapTriangle.valuesOffset ];

// find which block
blockIndex = computeBlockIndex( micromapTriangle.subdivisionLevel, micromapTriangle.blockFormat, microvertex.barycentricIntegerUV);
// BlockTriangle gives us information about the local position of the block within the base triangle, as well as byte offsets
BlockTriangle blockTriangle = computeBlockTriangle(micromapTriangle.subdivisionLevel, micromapTriangle.blockFormat, blockIndex );

ValueType uncompressedBlock[];

decompress(uncompressedBlock, micromapTriangle.subdivisionLevel, micromapTriangle.blockFormat, triangleValues + blockTriangle.byteOffset);


// use blockTriangle.vertexUVs[3] etc. to convert between map and block triangle coordinate space
ValueType microVertexValue = uncompressedBlock[ computeBlockUncompressedIndex(blockTriangle, microvertex.barycentricIntegerUV ) ];
```

Texture Data Flow



3D Mesh File





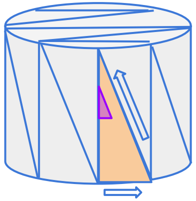
1. Mesh per-vertex uv-coordinates
2. UV-space triangle (winding invariant)
3. Texture sampling point/area
4. Filtered texture lookup

Texture File

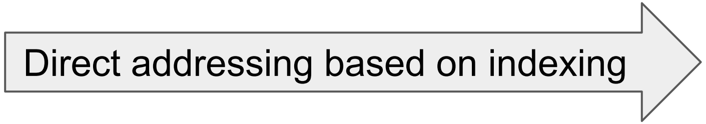
contains

- Grid of texels

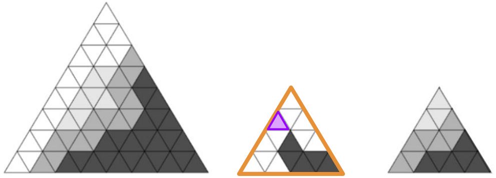
Micromap Uncompressed Data



3D Mesh File



1. Mesh per-triangle mapping index
2. Micromesh base triangle info
3. Microvertex or -triangle address (winding dependent)
4. Unfiltered data fetch



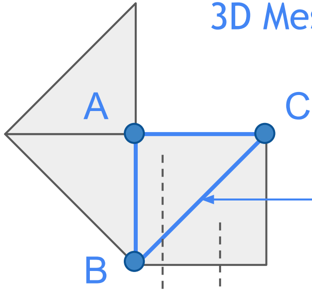
Barycentric File contains many triangles

- Buffer for base triangle information (subdiv level, values start offset)

- Buffer for all values (per micro-vertex or triangle)

Micromap Uncompressed Data

3D Mesh File



mesh.triangle

		1	2

mesh.triangleMappings buffer
- stores bary triangle index to instance the same bary values
- if omitted 1:1 mapping

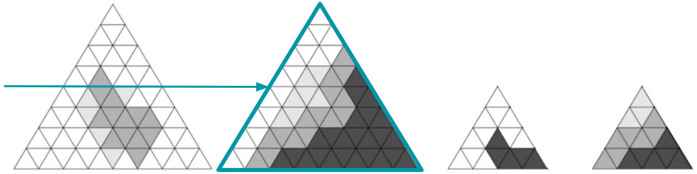
		2	0

mesh.triangleFlags buffer
- stores edge decimate flags

Triangle (A,B,C) = 2 (2nd bit set)
2nd edge (B,C) has half-res neighbor
and therefore needs decimation
(only relevant for displacement
water-tightness)

Barycentric File

bary.triangle
(e.g. subdiv 3,
64 microtris)



3	3	2	2
	..	..	

bary.triangle buffer
- subdivision level
- values start byte offset

bary.values buffer
- supports various data formats
- values within triangle ordered along spatial curve

Micromap Compressed Data

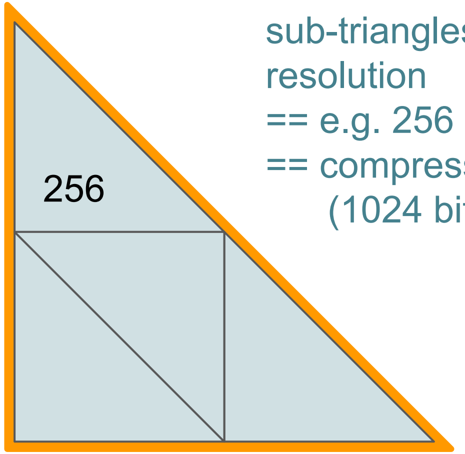
Displacement Compression

Compression may cause a split of the base-triangle into sub-triangles, each represented by a compressed block

Sub-triangles are ordered through the split logic and depth-first

A **single block compression format** is used within a base-triangle. However, currently three block formats exist, and so a file in total can use many formats.

Base triangle (e.g. 1024 microtris)
== developer choice



sub-triangles of **equal** resolution
== e.g. 256 microtris
== compressed block
(1024 bits)

Micromap Compressed Data

3D Mesh File

1. Mesh per-triangle mapping index
2. Micromesh base triangle info
3. Descend split-hierarchy & decompress appropriate data (winding dependent)
4. Unfiltered data use

Barycentric File contains many triangles

- Buffer for base triangle infos (subdiv level, compression format and blocks start offset)

- Buffer for compressed blocks

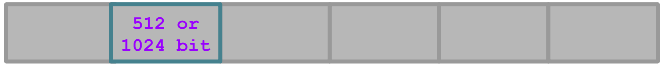
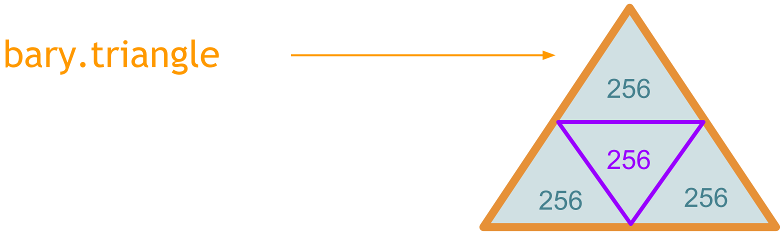
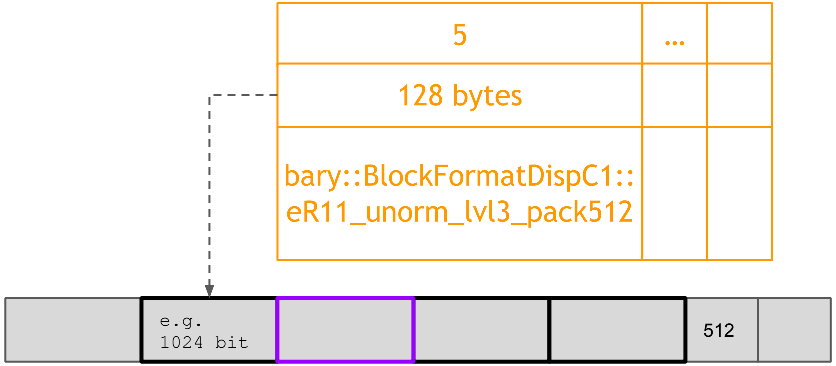


Illustration uses microtriangle values, though displacement is per microvertex

Micromesh Compressed Data



Split larger resolution into block-triangles.
These follow spatial, depth-first curve.



bary.triangle buffer

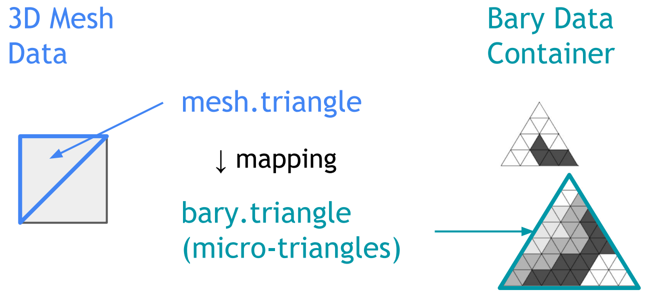
- subdivision level
- compressed blocks start byte offset
- compression format (implies subdivision level of a block and how many blocks exist)

bary.values buffer

- 512 and 1024 bit blocks
- Blocks for a single base triangle follow the “bird curve” ordering rules

Micromap Mesh Triangle Mapping

Triangle Mapping



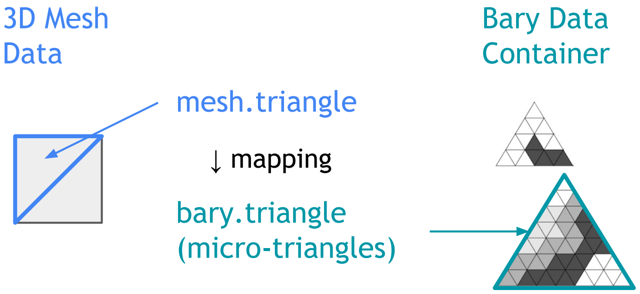
Most content will use a unique mapping, so that **each** mesh.triangle has its **own displacement** to capture unique detail.

However, in some occasions it might be useful to re-use displacement data.

For example to save memory when detail is cloned / instanced a lot within a bigger mesh (leaves in a tree, modular pieces creating a super-structure, …)

WARNING: not yet implemented in tools

UV to Micromap Index Idea



Mesh to micromap mapping is a per-mesh.triangle index

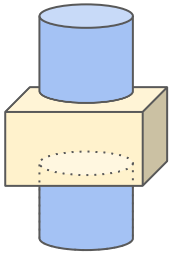
↯ Ugly to manage manually, editors use ngons…

↯ Dependency on triangle winding

Need something stable and ideally leveraging existing editor features

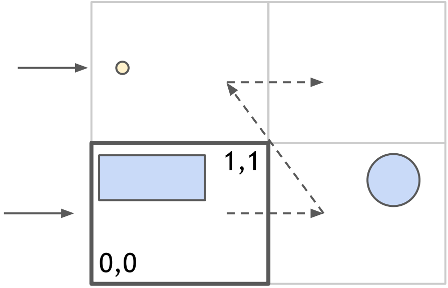
→ use 3 uv values of triangle as hash key to detect sharing of data or unique index gen.

→ use tri coordinates for deterministic value ordering in bary containers



Box uvs are degenerate dot, so it gets unique per-triangle mapping

Two cylinders re-use same uvs, so they will re-use mapping index



Sort explicit uvs along spatial curve within rectangle, then across rectangles (for sake of determinism only)

Additional Links & Information

These slides are part of the NVIDIA Micro-Mesh SDK

https://developer.nvidia.com/rtx/ray-tracing/micro-mesh

Relevant Repositories

https://github.com/NVIDIAGameWorks/Opacity-MicroMap-SDK
https://github.com/NVIDIAGameWorks/Displacement-MicroMap-SDK
https://github.com/NVIDIAGameWorks/Displacement-MicroMap-Toolkit
https://github.com/NVIDIAGameWorks/Displacement-MicroMap-BaryFile

Support Contacts for SDK

opacitymicromap-sdk-support@nvidia.com
DisplacedMicroMesh-SDK-support@nvidia.com