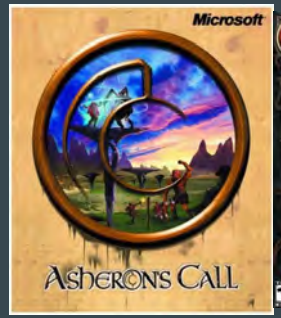
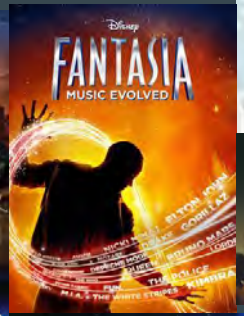
Behind the Assets

...

A technical deep dive into my assets, MicroSplat, MegaSplat, and One Batch





Jason Booth
3d Artist, Creative Director, Coder, Musician

Currently doing freelance contract coding, graphics and optimization

Optimization is about architecture

- The best optimizations are about a good choices in architecture

Optimization is about architecture

- The best optimizations are about a good choices in architecture
- CPU or GPU, it's mostly about memory, not algorithms
  - On CPU, muladd < 1 cycle. Cache miss, 700+ cycles
  - On GPU, bottlenecks can cause exponential performance drops
    - Especially when next stage is blocked
      - Vertex -> fragment
      - Dependent texture reads

Optimization is about architecture

- The best optimizations are about a good choices in architecture
- CPU or GPU, it's mostly about memory, not algorithms
  - On CPU, muladd < 1 cycle. Cache miss, 700+ cycles
  - On GPU, bottlenecks can cause exponential performance drops
    - Especially when next stage is blocked
      - Vertex -> fragment
      - Dependent texture reads
- New tech often driven by new optimization strategies

20 Assets in the Unity asset store

- MegaSplat
  - 256 texture splat mapping for Terrains and Meshes
- MicroSplat
  - Modular terrain shading system, broken into modules which can be individually purchased
- One Batch
  - Texture Array based atlasing for standard shader assets
- Stochastic Height Blending Node
  - For Unity's Shader Graph
  - For Amplify's Shader Graph

20 Assets in the Unity asset store

Common Tech themes

- Shader Generation Systems
  - Shader_feature too limiting (about 15 max features, want hundreds)
  - Rewrite shader code on demand based on features needed
    - No long build times due to shader variants!
  - MicroSplat add's interface based "modules" so that feature code can ship in different assets

20 Assets in the Unity asset store

Common Tech themes

- Shader Generation Systems
- Texture Arrays
  - Act as a single GPU resource
  - Users put in textures they have
    - Generate missing maps (normal, height, etc) from the data they give me
    - Pack into custom formats for speed or quality

20 Assets in the Unity asset store

Common Tech themes

- Shader Generation Systems
- Texture Arrays
- More performant than competition
  - With same feature spec, MicroSplat outperforms CTS by 2x-5x margin or more
  - One Batch allows hundreds of materials to be combined, retaining their unique settings

20 Assets in the Unity asset store

Common Tech themes

- Shader Generation Systems
- Texture Arrays
- More performant than competition
- Groundbreaking Features
  - Dynamically generated water and lava flows across terrains (MicroSplat)
  - Simple, one click terrain blending (MicroSplat)
  - 256 textures on one terrain or mesh with no performance penalty (MegaSplat)
  - Combining of hundreds of prefabs into a single material, supporting tiling and full texture resolution (One Batch)

MegaSplat

MegaSplat

- 256 textures on one terrain or mesh, with consistent texture sample count
  - First asset on the store

MegaSplat

- 256 textures on one terrain or mesh, with consistent texture sample count
  - First asset on the store
  - Some hard to maintain choices
    - vertex/fragment shader based

MegaSplat

- 256 textures on one terrain or mesh, with consistent texture sample count
  - First asset on the store
  - Some hard to maintain choices
    - vertex/fragment shader based
  - Feature set is too large
    - Vertex Painter, Terrain Painter, Texture Graph, Procedural workflows, etc, etc..

MegaSplat

- 256 textures on one terrain or mesh, with consistent texture sample count
  - First asset on the store
  - Some hard to maintain choices
    - vertex/fragment shader based
  - Feature set is too large
    - Vertex Painter, Terrain Painter, Texture Graph, Procedural workflows, etc, etc..
  - All-in mindset
    - Not designed for average Unity hobby user
    - Replaces entire terrain workflow with it's own

MegaSplat

- 256 textures on one terrain or mesh, with consistent texture sample count
  - First asset on the store
  - Some hard to maintain choices
    - vertex/fragment shader based
  - Feature set is too large
    - Vertex Painter, Terrain Painter, Texture Graph, Procedural workflows, etc, etc..
  - All-in mindset
    - Not designed for average Unity hobby user
    - Replaces entire terrain workflow with it's own
  - Doing minimal updates, but still sells well
    - $300-$800 a month
    - Unity featured heavily in the beginning

MegaSplat : Idea

- Store texture index into texture array in vertex

MegaSplat : Idea

- Store texture index into texture array in vertex
- Sample textures for each vertex in pixel shader
  - 3 vertices per triangle, 3 samples

MegaSplat : Idea

- Store texture index into texture array in vertex
- Sample textures for each vertex in pixel shader
  - 3 vertices per triangle, 3 samples
- Blend results

MegaSplat : Idea

- Store texture index into texture array in vertex
- Sample textures for each vertex in pixel shader
  - 3 vertices per triangle, 3 samples
- Blend results
- Texture arrays can have 1024 textures,
  so splat mapping with up to 1024 textures!

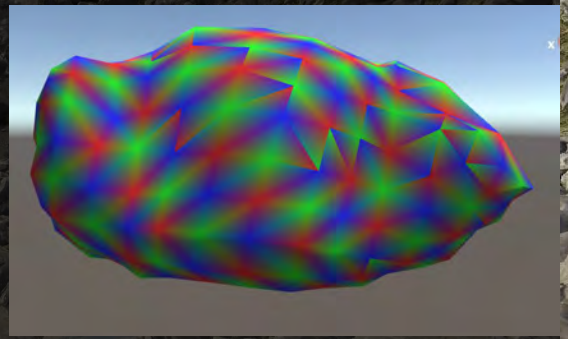
MegaSplat : Idea

- Store texture index into texture array in vertex
- Sample textures for each vertex in pixel shader
  - 3 vertices per triangle, 3 samples
- Blend results
- Texture arrays can have 1024 textures,
  so splat mapping with up to 1024 textures!

Problem: Interpolation of vertex data

MegaSplat : Solution

- Use Barycentric coordinates to reconstruct data
  - Mark each triangle with one red, one blue, and one green vertex
  - Preprocess mesh, or in Geometry Shader

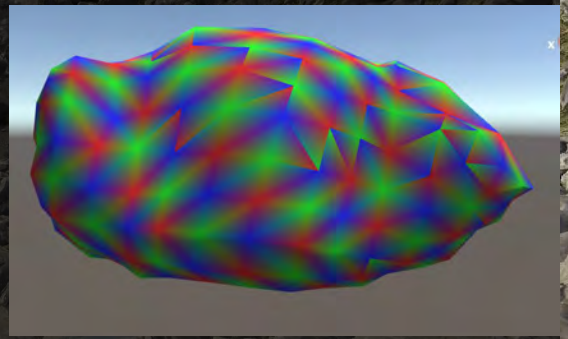


MegaSplat : Solution

- Use Barycentric coordinates to reconstruct data
  - Mark each triangle with one red, one blue, and one green vertex
  - Preprocess mesh, or in Geometry Shader

Encode (in vertex shader):

o.values.xyz = i.color.rgb * index;



MegaSplat : Solution

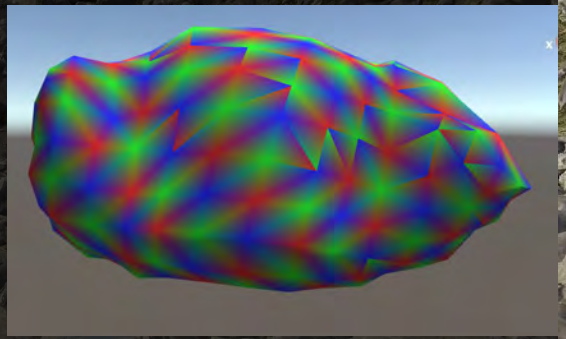
- Use Barycentric coordinates to reconstruct data
  - Mark each triangle with one red, one blue, and one green vertex
  - Preprocess mesh, or in Geometry Shader

Encode (in vertex shader):

```
o.values.xyz = i.color.rgb * index;
```

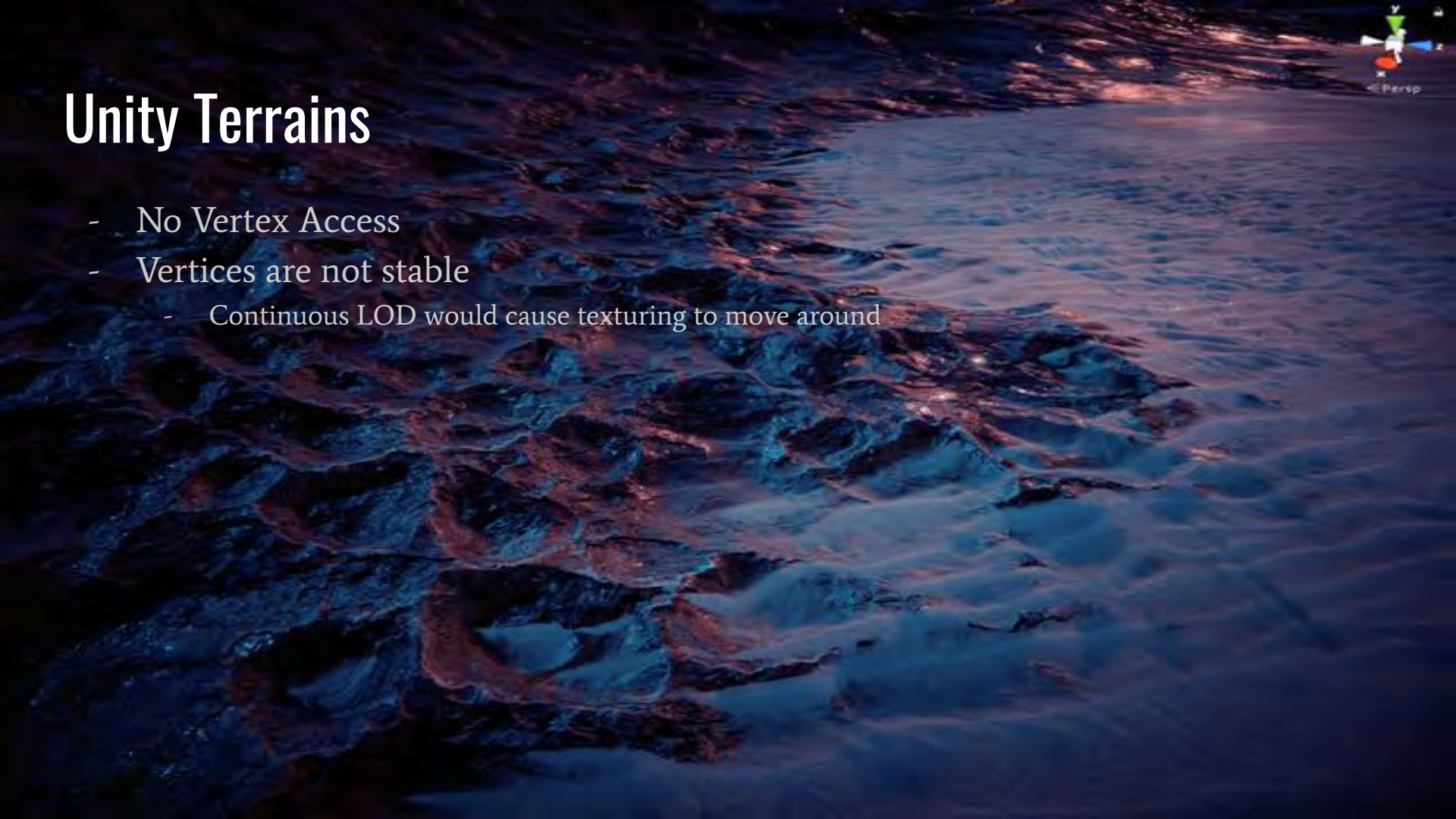
Decode (in fragment shader):

```
int i0 = round(o.values.x / max(o.color.x, 0.00001));
int i1 = round(o.values.y / max(o.color.y, 0.00001));
int i2 = round(o.values.z / max(o.color.z, 0.00001));
```



Unity Terrains

- No Vertex Access
- Vertices are not stable
  - Continuous LOD would cause texturing to move around

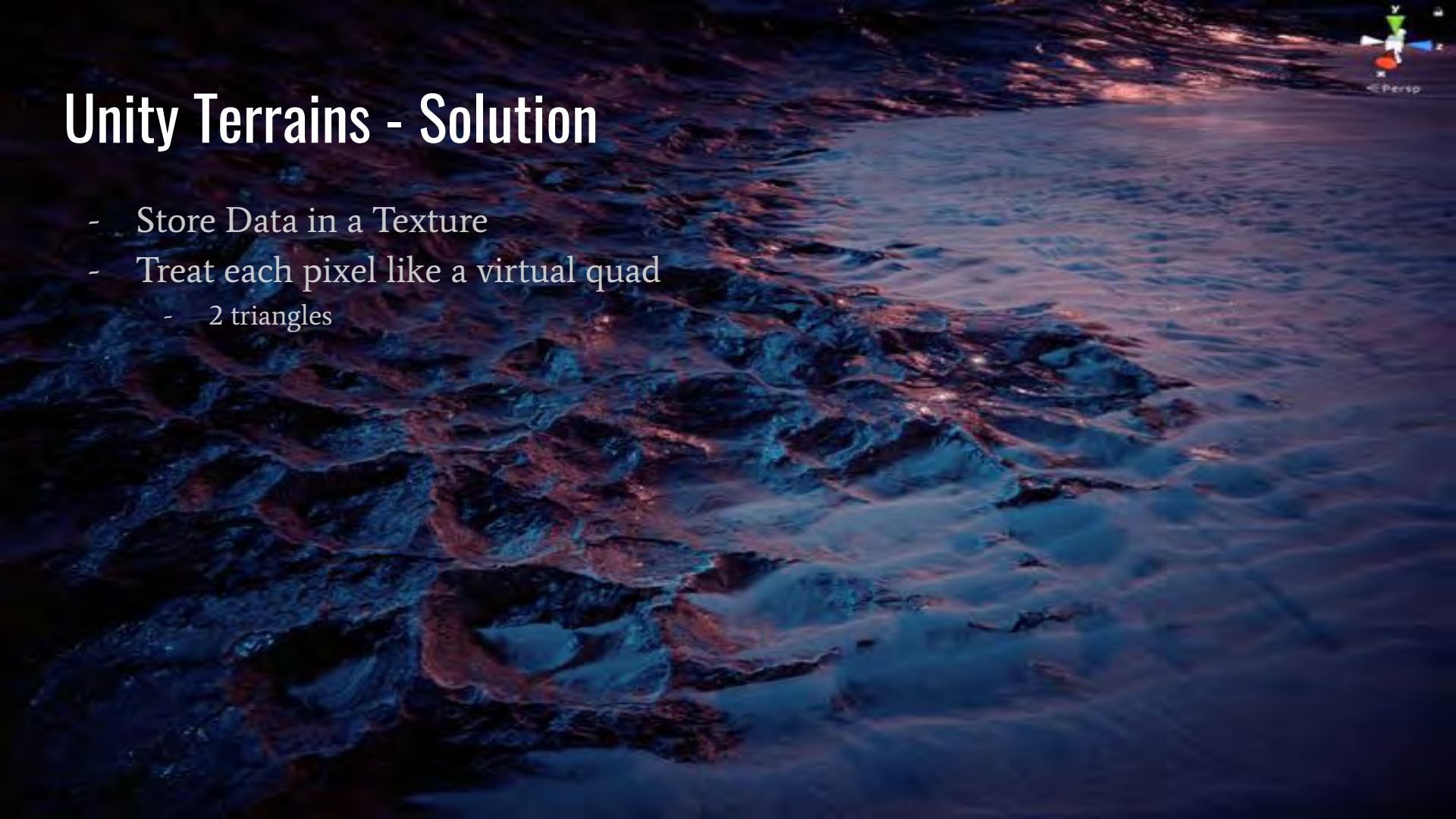


Unity Terrains - Solution

- Store Data in a Texture

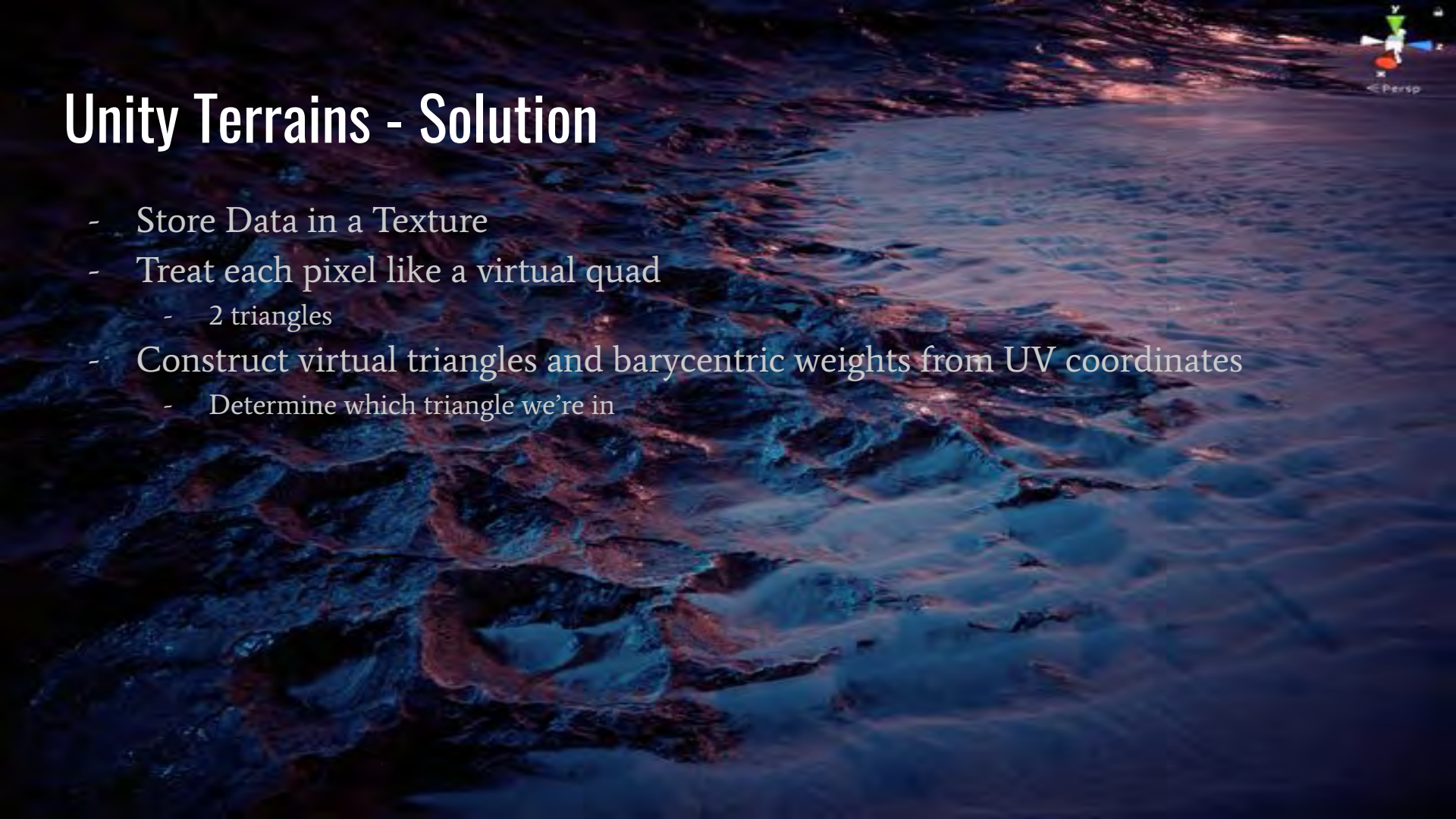
Unity Terrains - Solution

- Store Data in a Texture
- Treat each pixel like a virtual quad
  - 2 triangles



Unity Terrains - Solution

- Store Data in a Texture
- Treat each pixel like a virtual quad
  - 2 triangles
- Construct virtual triangles and barycentric weights from UV coordinates
  - Determine which triangle we're in



Unity Terrains - Solution

- Store Data in a Texture
- Treat each pixel like a virtual quad
  - 2 triangles
- Construct virtual triangles and barycentric weights from UV coordinates
  - Determine which triangle we're in
- Sample 3 indexes, textures, blend

Practical limitations

- One texture at each vertex
  - Must be 100% weighted at that vertex
  - This means blending area == triangle area
    - Doesn't always look great
    - Lower res control maps look better, but less control
- Blending is barycentric
  - 3 point blend can create zig zag effect
  - Soft blends often desirable

Practical Solutions

- Multiple ‘layers’
  - Encode second texture index in UV channel
  - Encode blend weight
  - Soft, controllable blends, but only between layers
  - Understanding layers can be confusing
- Tooling
  - ‘AutoBrush’ paints on lowest weighted layer
    - Makes best choice for you

Texture Clustering

- What do you do with so many textures?

Texture Clustering

- What do you do with so many textures?
- If each vertex can have it's own texture, why not vary the choice per vertex?
  - Choose from a pallet of texture variations
  - Choose based on noise, slope, height

Texture Clustering

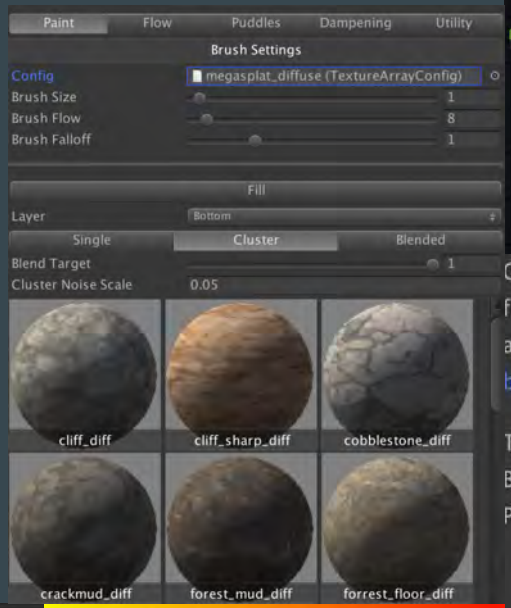
- What do you do with so many textures?
- If each vertex can have it's own texture, why not vary the choice per vertex?
  - Choose from a pallet of texture variations
  - Choose based on noise, slope, height
- Texture Clustering "first level concept" in MegaSplat

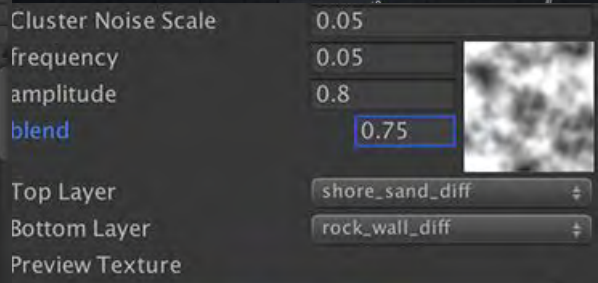
Woe of blending

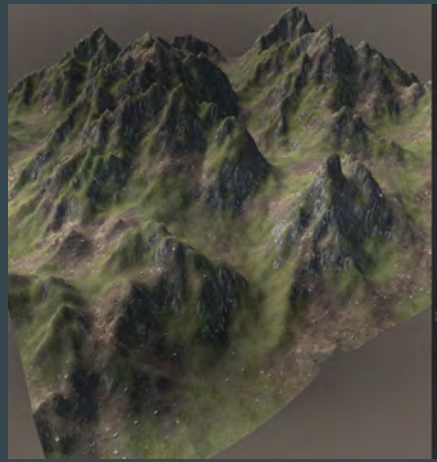
- Lots of research into how to make terrains look good with this technique
  - Texture Clustering as an expectation
    - Use substance/Mixer to produce variations on a theme
    - Mix several complementary surfaces with procedurals to create a meta-surface
  - Multi-Layers
    - Second index and blend weight
  - Auto Brush
    - Automatically selects the layer with the least weight to paint on
  - Texture Graph
    - Procedural, graph based texture tool
  - Terrain conversion tools
    - Map original texture -> cluster
  - Ideal ratio of vertex density to texture blends

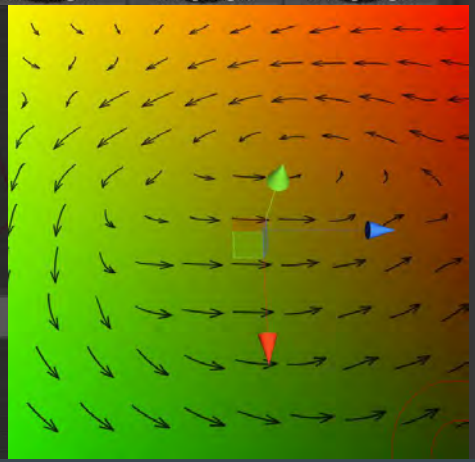
Highly customized tools

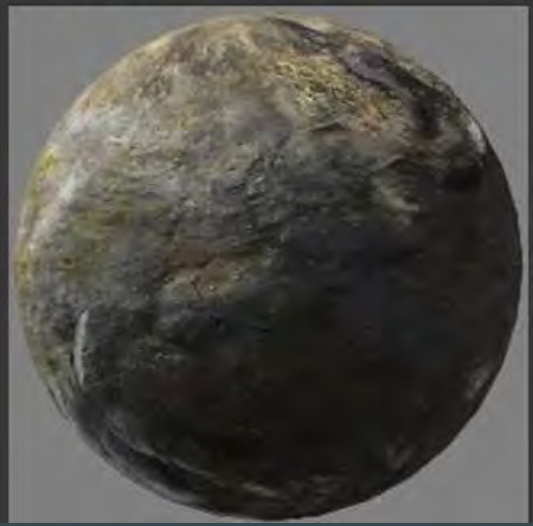
Graph based, painting, converting, previewing, etc..











MegaSplat

- MegaPackage
  - Hard for someone to evaluate what it does/doesn't do
  - Lots to maintain
    - Mesh vs. Terrain, Tessellated vs. Non tessellated, Vertex/Fragment shader
    - Lots of tools
    - Compatibility with other tools
  - What is actually being used?
  - One sale and done

MicroSplat

...


*Assets not included
FALCON AGE

MicroSplat - Goals

- CTS: Can you tell us why our shader is slow?
  - Yes!
  - How would I do this?

MicroSplat - Goals

- CTS: Can you tell us why our shader is slow?
  - Yes!
  - How would I do this?
- Wanted something more:
  - Standard
  - Maintainable
  - Modular, expandable
  - Limited scope (terrain only, originally)
  - Easier to use for average user
  - Higher total price point and easy to evaluate

MicroSplat

- Modular terrain shader
  - Core module is free
  - Feature modules sold from $5 to $25 each
  - 16 modules currently
    - One module written by a 3rd party

MicroSplat

- Modular terrain shader
  - Core module is free
  - Feature modules sold from $5 to $25 each
  - 16 modules currently
    - One module written by a 3rd party
- Pay for the features you need
  - Sales give me analytics on feature usage
  - Often have half of the terrain section filled with my modules
  - Can upsell from free
    - 1200 downloads of free version each month

MicroSplat

- Modular terrain shader
  - Core module is free
  - Feature modules sold from $5 to $25 each
  - 16 modules currently
    - One module written by a 3rd party
- Pay for the features you need
  - Sales give me analytics on feature usage
  - Often have half of the terrain section filled with my modules
  - Can upsell from free
    - 1200 downloads of free version each month
- Focused on ease of use
  - Simple to begin, gets more complex as you add modules

MicroSplat Modules

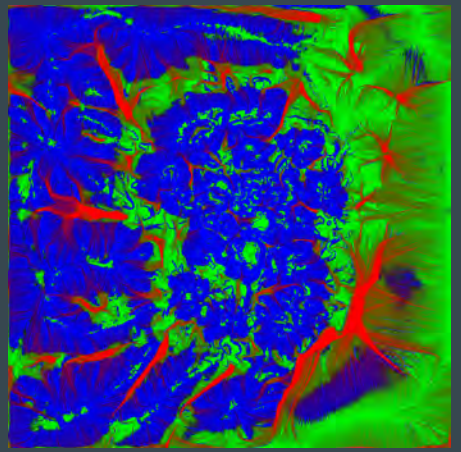
- Advanced Details (3rd party)
- AntiTile
- Global Texturing
- Lightweight Render Pipeline
- HDRP
- Mesh Terrains
- Mesh Workflow
- Runtime Procedural Texturing
- Snow

- Streams, Lava, Wetness, Puddles
- Terrain Blending
- Terrain Holes
- Tessellation & Parallax
- Texture Clusters
- Triplanar
- Wind and Glitter


MICROSPLAT
TERRAIN BLENDING

Unity Terrain

- Uses an RGBA control mask for every 4 textures on the terrain. Each channel represents a weight for that texture. This is normalized across all weights.

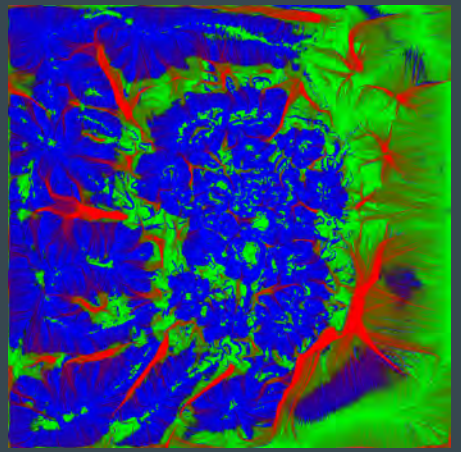


Unity Terrain

- Uses an RGBA control mask for every 4 textures on the terrain. Each channel represents a weight for that texture. This is normalized across all weights.

Unity Terrain Shader:

- Render terrain once for every 4 textures used.

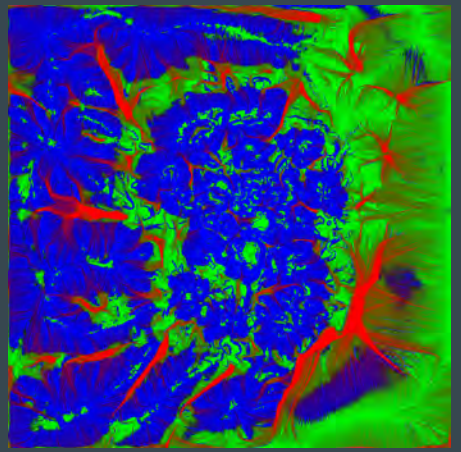


Unity Terrain

- Uses an RGBA control mask for every 4 textures on the terrain. Each channel represents a weight for that texture. This is normalized across all weights.

Unity Terrain Shader:

- Render terrain once for every 4 textures used.
- Sample 4 albedo, 4 normals, and linear blend together

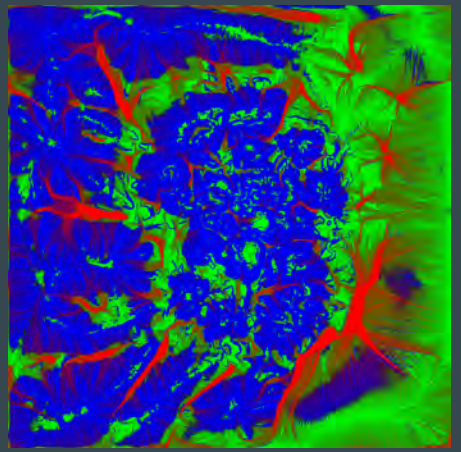


Unity Terrain

- Uses an RGBA control mask for every 4 textures on the terrain. Each channel represents a weight for that texture. This is normalized across all weights.

Unity Terrain Shader:

- Render terrain once for every 4 textures used.
- Sample 4 albedo, 4 normals, and linear blend together
- Blend with result in frame buffer

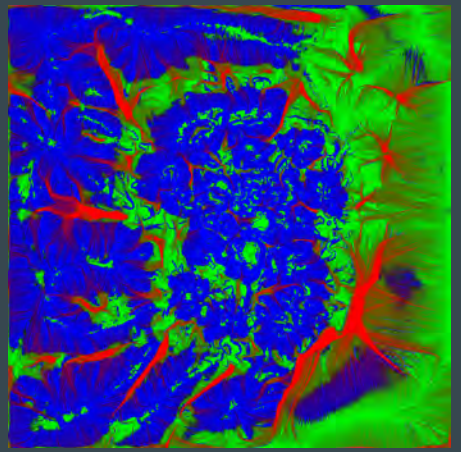


Unity Terrain

- Uses an RGBA control mask for every 4 textures on the terrain. Each channel represents a weight for that texture. This is normalized across all weights.

Unity Terrain Shader:

- Render terrain once for every 4 textures used.
- Sample 4 albedo, 4 normals, and linear blend together
- Blend with result in frame buffer
- 16 textures = 16 albedo + 16 normal + 4 control samples
  Draw the terrain 4 times



Unity Terrain

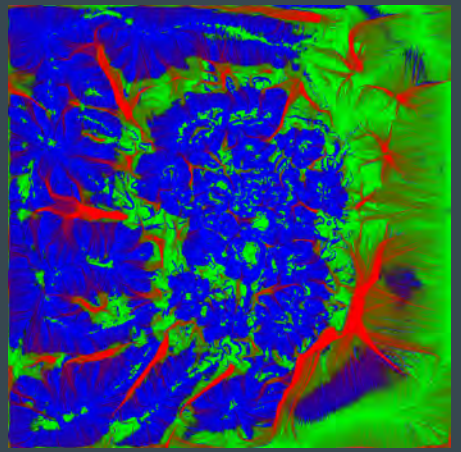
- Uses an RGBA control mask for every 4 textures on the terrain. Each channel represents a weight for that texture. This is normalized across all weights.

Unity Terrain Shader:

- Render terrain once for every 4 textures used.
- Sample 4 albedo, 4 normals, and linear blend together
- Blend with result in frame buffer
- 16 textures = 16 albedo + 16 normal + 4 control samples
  Draw the terrain 4 times
- Simple, but inefficient and looks bad
  - Normals don't even blend right after 8 textures..

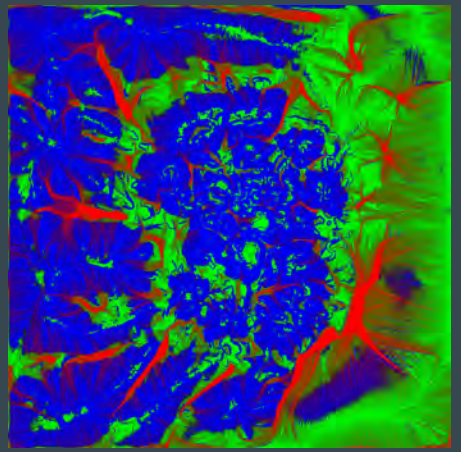
CTS - why so slow?

- Natural extension of Unity system (though single pass)
  - Textures packed into 3 arrays (diffuse+height, normal, smoothness/AO)
  - 16 textures = 48 samples + 4 control maps



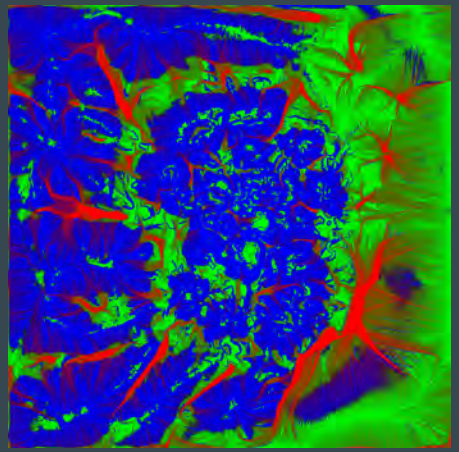
CTS - why so slow?

- Natural extension of Unity system (though single pass)
  - Textures packed into 3 arrays (diffuse+height, normal, smoothness/AO)
  - 16 textures = 48 samples + 4 control maps
- Triplanar
  - Sample each texture from a top, front, side projection
  - 3x samples on albedo/normal, so 48*3 + 4 control



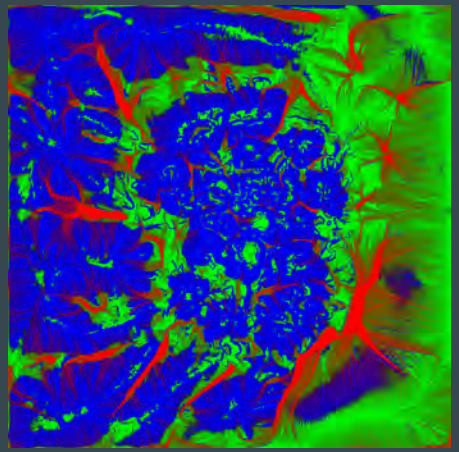
CTS - why so slow?

- Natural extension of Unity system (though single pass)
  - Textures packed into 3 arrays (diffuse+height, normal, smoothness/AO)
  - 16 textures = 48 samples + 4 control maps
- Triplanar
  - Sample each texture from a top, front, side projection
  - 3x samples on albedo/normal, so 48*3 + 4 control
- Distance Resampling
  - 2x sample count, so 48*3*2 + 4 control (292 samples)



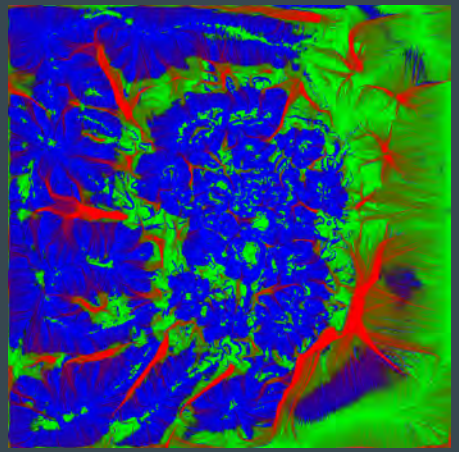
CTS - why so slow?

- Natural extension of Unity system (though single pass)
  - Textures packed into 3 arrays (diffuse+height, normal, smoothness/AO)
  - 16 textures = 48 samples + 4 control maps
- Triplanar
  - Sample each texture from a top, front, side projection
  - 3x samples on albedo/normal, so 48*3 + 4 control
- Distance Resampling
  - 2x sample count, so 48*3*2 + 4 control (292 samples)
- No feature culling
  - Runs full triplanar when not in use
  - Other features push sample count even higher



CTS - why so slow?

- Natural extension of Unity system (though single pass)
  - Textures packed into 3 arrays (diffuse+height, normal, smoothness/AO)
  - 16 textures = 48 samples + 4 control maps
- Triplanar
  - Sample each texture from a top, front, side projection
  - 3x samples on albedo/normal, so 48*3 + 4 control
- Distance Resampling
  - 2x sample count, so 48*3*2 + 4 control (292 samples)
- No feature culling
  - Runs full triplanar when not in use
  - Other features push sample count even higher
- Extraneous math
  - Each normal decoded and unpacked at sample time.
  - Shader graphs hide complexity and potential optimizations



MicroSplat - Why so fast?

- More than a few textures are ever visible for a single pixel, so why sample them all?

MicroSplat - Why so fast?

- More than a few textures are ever visible for a single pixel, so why sample them all?
- Branches around texture samples break 2x2 block concurrency! 4x slower!

MicroSplat - Why so fast?

- More than a few textures are ever visible for a single pixel, so why sample them all?
- Branches around texture samples break 2x2 block concurrency! 4x slower!
- Idea:
  - Sample control maps
  - Sort weights/indexes
  - Only sample fixed number of textures
    - This can be adjusted by the user for performance/quality tradeoff
    - Can use same trick in techniques that don't need as much fidelity (distance resampling)

MicroSplat - consistent sample counts FTW

- Default packing of textures in 2 arrays (can do 3 if you want)
  - Albedo + Height
  - Smoothness, Normal G, AO, Normal R
    - Normal in G/A gives best quality for compression
    - Compressors favor green, because the eye is more sensitive to green than other colors
    - Alpha channel is compressed separately from RGB

MicroSplat - consistent sample counts FTW

- Default packing of textures in 2 arrays (can do 3 if you want)
  - Albedo + Height
  - Smoothness, Normal G, AO, Normal R
    - Normal in G/A gives best quality for compression
    - Compressors favor green, because the eye is more sensitive to green than other colors
    - Alpha channel is compressed separately from RGB
- Only sample 4 most prominent texture sets
  - For 16 textures, 4 control maps, 4 albedo, 4 normal samples = 12 samples
  - Triplanar : 28 samples
  - Distance Resampling
    - Only perform on top 2 textures
    - Fast: Only do albedo, 2 additional samples (6 if triplanar)
    - Full: Albedo and Normal, 4 additional samples (12 if triplanar)

Compare

CTS, 16 texture terrain:

- Base: 48 samples + 4 control maps

MicroSplat, 32 texture terrain

- Base: 8 samples + 8 control maps

Compare

CTS, 16 texture terrain:

- Base: 48 samples + 4 control maps
- Triplanar 144 samples + 4 control maps

MicroSplat, 32 texture terrain

- Base: 8 samples + 8 control maps
- Triplanar: 24 samples + 8 control maps

Compare

CTS, 16 texture terrain:

- Base: 48 samples + 4 control maps
- Triplanar 144 samples + 4 control maps
- Distance Resample 288 samples + 4 control

MicroSplat, 32 texture terrain

- Base: 8 samples + 8 control maps
- Triplanar: 24 samples + 8 control maps
- Distance Resample 36 samples + 8 control

Compare

CTS, 16 texture terrain:

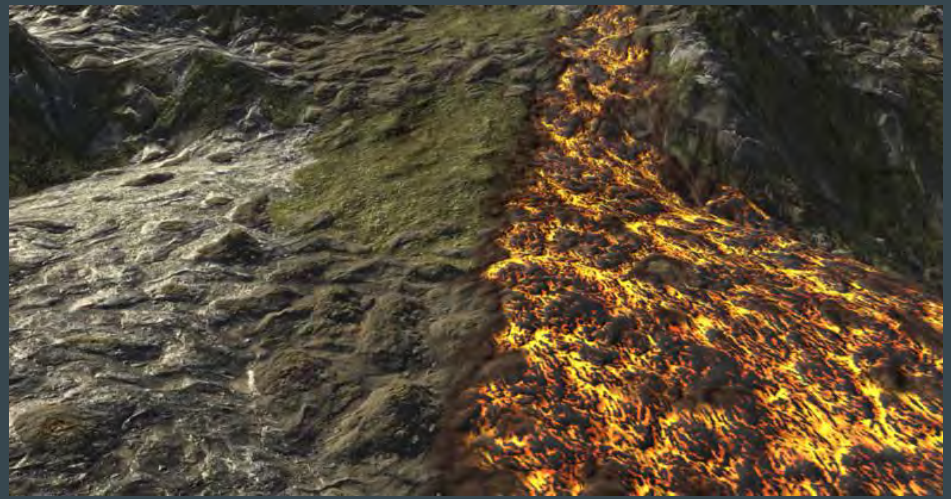
- Base: 48 samples + 4 control maps
- Triplanar 144 samples + 4 control maps
- Distance Resample 288 samples + 4 control

MicroSplat, 32 texture terrain

- Base: 8 samples + 8 control maps
- Triplanar: 24 samples + 8 control maps
- Distance Resample 36 samples + 8 control

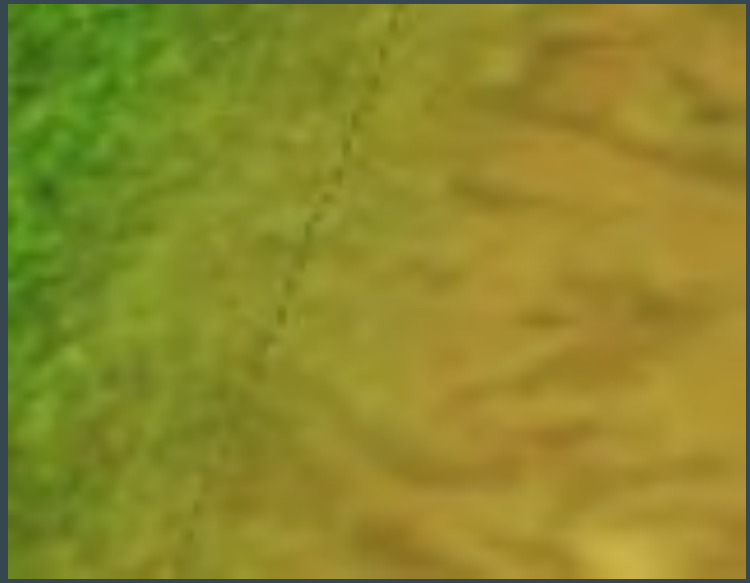
Which leaves tons of room for other features, like:

- Real time streams and Lava, flowing from emission controllers
- Automatic Terrain Blending with scene objects
- Wind and glitter effects
- Texture Clustering (96 textures on terrain)



No good deed goes unpunished

- A fantastic optimization
  - But errors when per texture UV scale is used
  - GPU's don't like it
  - Small seams between blending areas discovered
- WTF?
  - Took a while to figure out what was happening..
  - Texture Derivative issue with the way GPU's sample textures

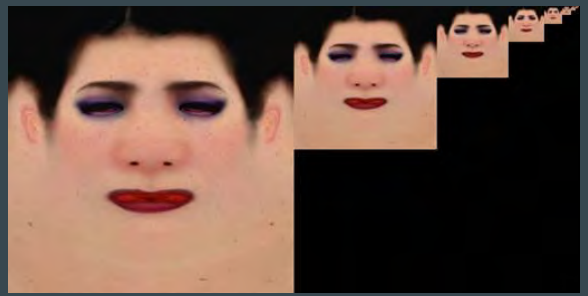


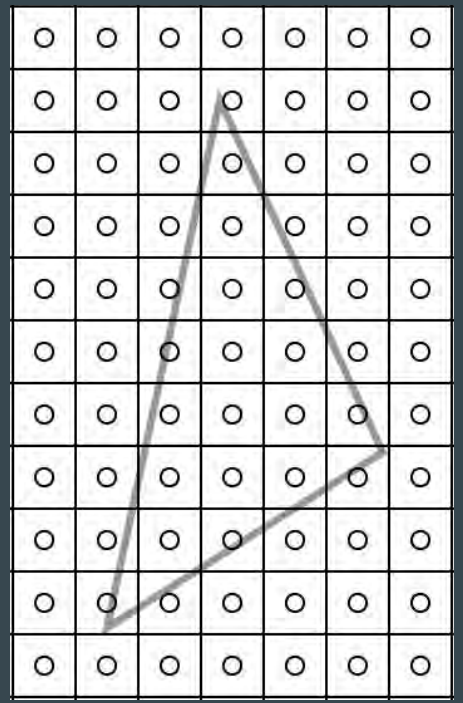
To understand, a deep dive into how GPUs work...

Texture Derivatives

As a triangle gets rasterized on screen:

- It needs to determine the mip-map's to sample from
  - "Based on distance from camera!" - NO
  - Based on texture size projected into screen space

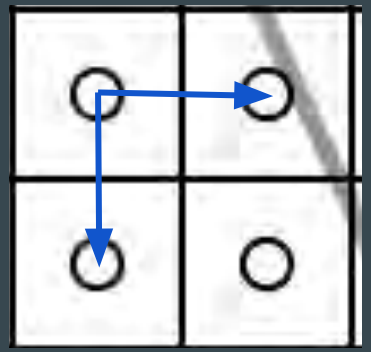


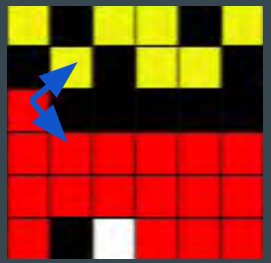


Texture Derivatives

As a triangle gets rasterized on screen:

- ddx(uv)
  - Vector pointing to next pixel in texel space on X axis
- ddy(uv)
  - Vector pointing to next pixel in texel space on Y axis
- Bilinear filtering
  - Look up 4 sample points, blend between them
- Trilinear
  - Look up 4 sample points at 2 different mip levels, blend between them

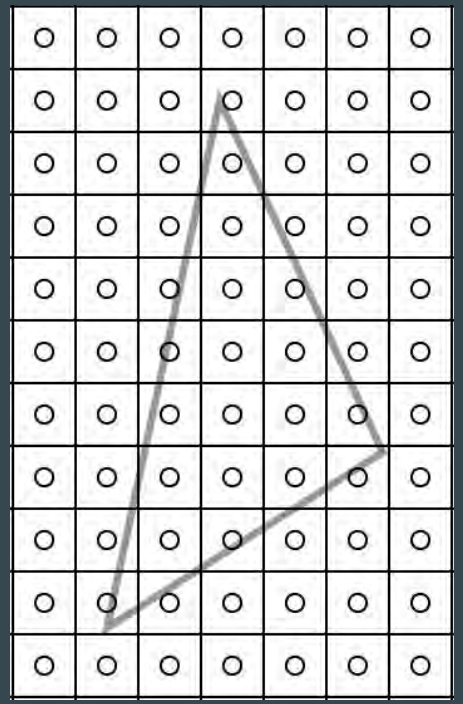




Texture Derivatives

GPU optimization

- When computing pixels, compute them in 2x2 blocks
  - You need to look up 4 pixels at once to compute bilinear blend
  - Fetching texture data can be shared between pixels!
    - Massive speedup in shading rate!
    - This is why Mip Maps are important : Cache Coherency



The Bug

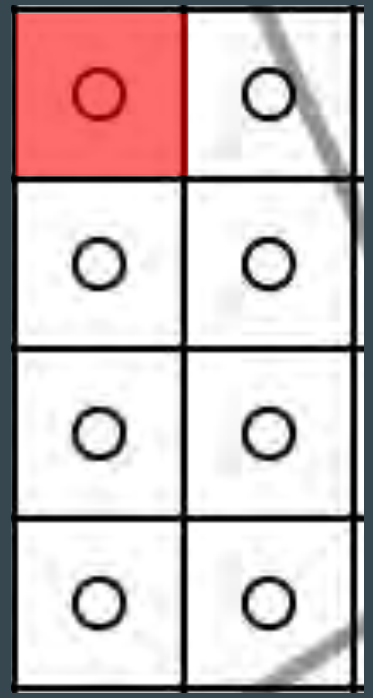
We sort textures based on weight:

0.49 weight, uv scale 1.5

0.43 weight, uv scale 0.8

0.07 weight, uv scale 1.1

0.01 weight, uv scale 0.8



The Bug

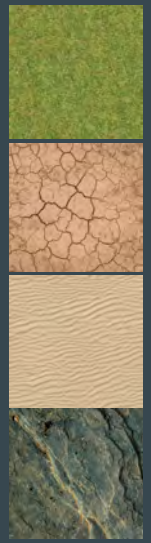
We sort textures based on weight:

0.49 weight, uv scale 1.5

0.43 weight, uv scale 0.8

0.07 weight, uv scale 1.1

0.01 weight, uv scale 0.8

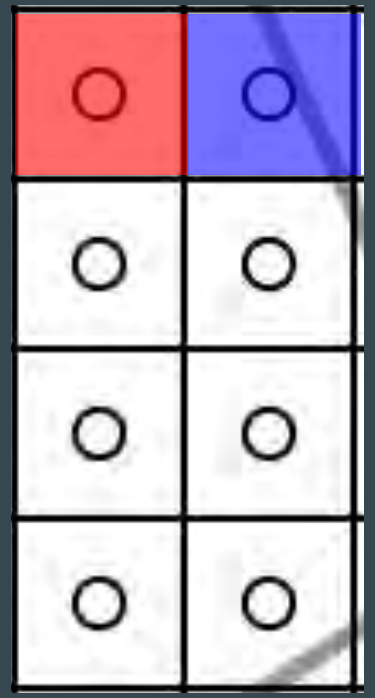


0.49 weight, uv scale 0.8

0.43 weight, uv scale 1.5

0.07 weight, uv scale 1.1

0.01 weight, uv scale 0.8



The Bug

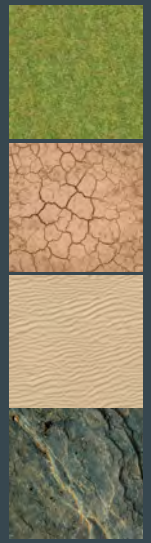
We sort textures based on weight:

0.49 weight, uv scale 1.5

0.43 weight, uv scale 0.8

0.07 weight, uv scale 1.1

0.01 weight, uv scale 0.8

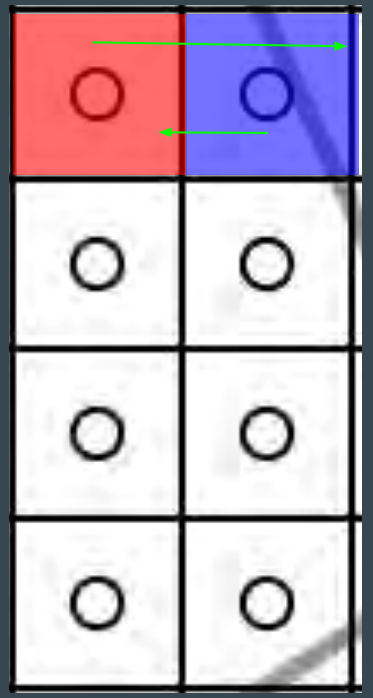


0.49 weight, uv scale 0.8

0.43 weight, uv scale 1.5

0.07 weight, uv scale 1.1

0.01 weight, uv scale 0.8



Pixels derivatives do not match! Mip mapping will be incorrect!

Options?

- Use LOD samplers, where you specify the LOD manually
  - Which LOD?
  - Breaks trilinear filtering and anisotropic filtering
  - Run's faster than using regular samplers!?

Options?

- Use LOD samplers, where you specify the LOD manually
  - Which LOD?
  - Breaks trilinear filtering and anisotropic filtering
  - Run's faster than using regular samplers!?
- Gradient Samplers
  - Gradient samplers let you supply your own derivatives rather than letting the GPU compute them
  - How to compute? ddx(uv) will return different result for both pixels!

Options?

- Use LOD samplers, where you specify the LOD manually
  - Which LOD?
  - Breaks trilinear filtering and anisotropic filtering
  - Run's faster than using regular samplers!?
- Gradient Samplers
  - Gradient samplers let you supply your own derivatives rather than letting the GPU compute them
  - How to compute? ddx(uv) will return different result for both pixels!
- Break 2x2 optimization
  - Too slow!

Answer

- Get derivatives from unscaled UVs
- When UV scale is applied, also scale derivatives
- Average derivative per pixel based on layer weights

```
float4 dxdy = float4(ddx(uv), ddy(uv));
dxdy = dxdy * uvScale[0] * weights[0] +
       dxdy * uvScale[1] * weights[1] +
       dxdy * uvScale[2] * weights[2] +
       dxdy * uvScale[3] * weights[3];
```

- Closen mip level is interpolated across boundary, such that all 4 pixels get the same result.

And on it goes..

Triplanar Mapping:

- Compute UVs for each projection from world position
- Compute derivatives for each projection from world position
- Apply average scaling, etc.

```
float3 dxw = ddx(worldPos);
float3 dyw = ddy(worldPos);
float2 uvXdxdy = float4(dxw.zy, dyw.zy);
float2 uvYdxdy = float4(dxw.xz, dyw.xz);
float2 uvZdxdy = float4(dxw.xy, dyw.xy);
```

But one example

Other techniques:

- Keep normals packed (2 channel) through entire pipeline and decode at the end
- Sorting routine profiled and rewritten many times
  - Some that I expected would be fast were actually really slow! Always profile
  - Only sort when > 4 textures (low end optimization)
- Per texture property data stored in lookup texture (32x32, 16 bit)
  - No support for array's of properties anyway
  - Setting/lookup of property data slower than texture fetch!
- Every unused option compiled out
- Distance terrain uses a simpler shader

Modular Shaders

- FeatureDescriptor
  - GUI for turning features on/off
  - GUI for adjusting shader values
  - GUI for adjusting per-texture properties
  - Pack/Unpack serialization functions
  - Function to report shader cost
  - Write Properties, cbuffers, etc
  - Write functions

Modular Shaders

- FeatureDescriptor
  - GUI for turning features on/off
  - GUI for adjusting shader values
  - GUI for adjusting per-texture properties
  - Pack/Unpack serialization functions
  - Function to report shader cost
  - Write Properties cbuffers, etc
  - Write functions
- Shader compiler
  - IRenderAdapter for adapting different wrappers around shaders
    - Used for Surface Shader vs. SRP shaders

Modular Shaders

- Keyword storage system
  - Uses scriptable object instead of material keywords
    - Unity has a limit of 256 per project
      - MicroSplat would use over 700
    - MicroSplat uses no actual Unity keywords

Modular Shaders

- Keyword storage system
  - Uses scriptable object instead of material keywords
    - Unity has a limit of 256 per project
      - MicroSplat would use over 700
    - MicroSplat uses no actual Unity keywords
- PropData object
  - Stores per-texture property data, which gets converted into a texture at runtime
    - Unity cannot correctly serialize RGBAHalf or RGBAFloat linear textures

Modular Shaders

- Keyword storage system
  - Uses scriptable object instead of material keywords
    - Unity has a limit of 256 per project
      - MicroSplat would use over 700
    - MicroSplat uses no actual Unity keywords
- PropData object
  - Stores per-texture property data, which gets converted into a texture at runtime
    - Unity cannot correctly serialize RGBAHalf or RGBAFloat linear textures
- Core module has “master” shader code chunk
  - Calls all module functions with #if _SOMEFEATURE around calls
  - Core module must be in sync with external modules
    - Locking mechanism to skip modules which are not the current version

Modular Shaders

- Compiler
  - Writes header/properties
  - For each pass in IRenderLoopAdapter
    - Writes pass header
    - Writes properties
    - Writes #define keywords for enabled features
      - #define _SOMEFEATURE 1
    - Writes functions
    - Writes master shader

Modular Shaders

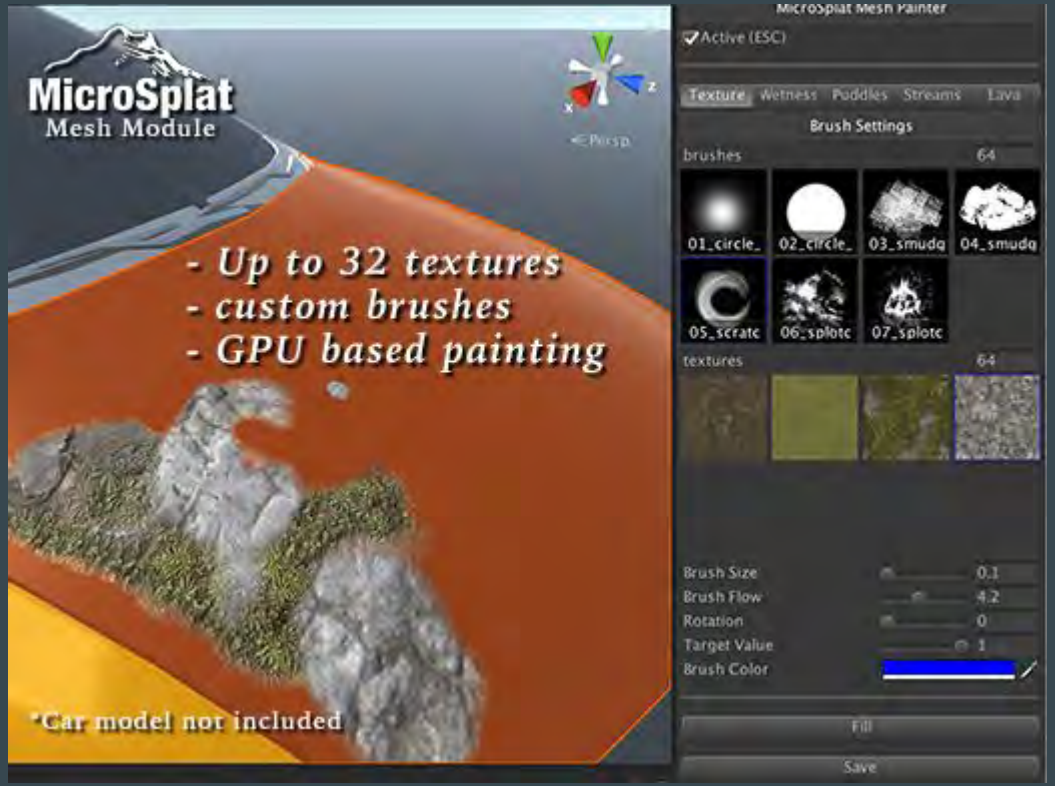
- Compiler
  - Writes header/properties
  - For each pass in IRenderLoopAdapter
    - Writes pass header
    - Writes properties
    - Writes #define keywords for enabled features
      - #define _SOMEFEATURE 1
    - Writes functions
    - Writes master shader
- Shaders are rewritten any time features are changed
  - Thus any unused feature is not in the actual code

Asset Store

- Regularly has 4-8 modules on the front page of terrain
- Free core module downloaded 1000 to 2000 times a month
  - Modules advertised in core module
  - Extra $50-$80 a month earned via affiliate program
- $1000-$3000 a month in sales (after unity's 30%)
  - Best month $6500
- Never featured by Unity
  - 95/98 written reviews are 5 star
  - On the 'most popular free assets' list every month
- Terrain Collection bundle released
  - More likely to be included in sales, which often have minimum pricing
  - Perfect 5 star rating (14 written reviews)
- Total price for all modules $224
  - But each one seems cheap for what you get

Coming soon

- Mesh workflows
  - Uses MicroSplat shader compiler back end, but with mesh painting
  - Can be blended with existing shaders
- Deformation module
  - Snow tracks
  - “Spin Tires” style games



Downsides

- Updating all modules with a version change is a pain
  - For me, and for users
- Not as “One Click” as could be
  - Having all features included means many can default to ON
  - Some people compare free version to $50 asset
- HDRP/LWRP/URP support is a nightmare
  - Unity constantly changing these pipelines
  - No abstraction layer
  - No automatic upgrading
  - No documentation

One Batch

One Batch

- Currently in Beta
- Idea came from MegaSplat users
  - House made of many separate pieces, thus multiple draw calls
  - Put all textures into texture arrays
  - Use painter to assign textures fully to each vertex
    - Door, wall, floor, etc..
  - Combine mesh
    - 1 material, 1 draw call
    - Can combine entire scene into one material
- Why not formalize into easier process?

One Batch

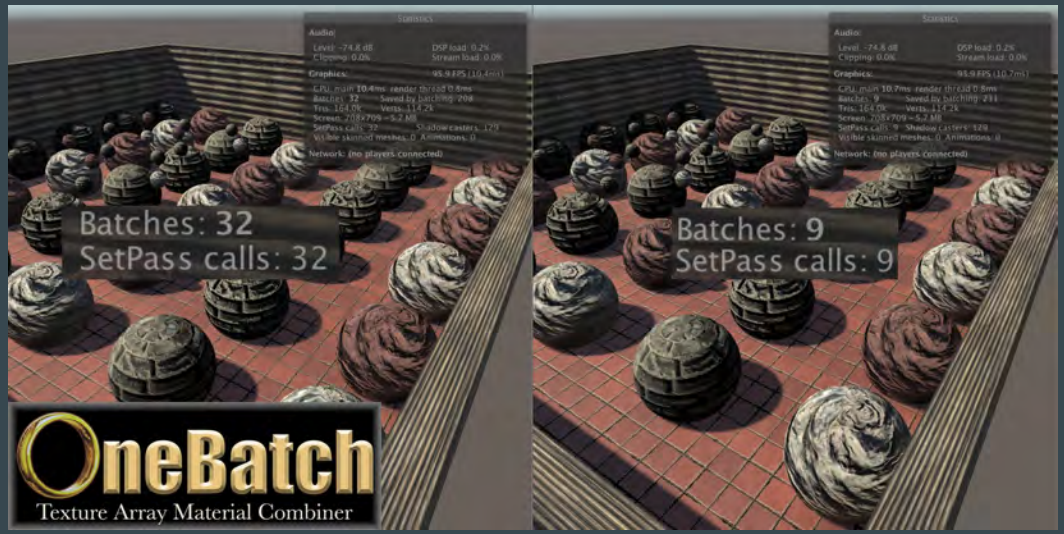
- From a list of prefabs:
  - Automatically extract all textures and material properties
  - Pack textures into arrays
  - Pack material properties into texture look up table
  - Store index for texture array and index for properties in mesh color channels
  - Combine all submeshes of an object
  - Assign single One Batch material and shader to everything
- Limitations
  - Currently tied to the standard shader only
    - No automatically convert user shaders to use texture arrays and a property LUT
  - Designed as a ‘late stage optimizations’
    - Take this kit bashed level and combine it
    - Can still edit material data after the fact, but easier to uncombine, edit, then recombine

Excellent results, in the right situation

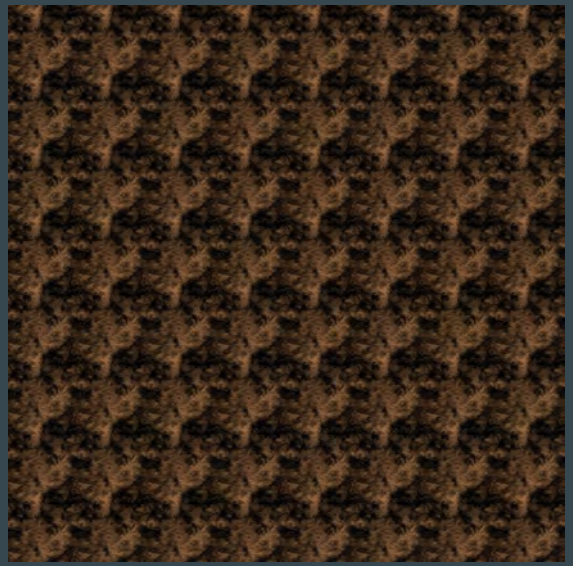
- Many things break static batching in Unity (lighting!)
- Already optimized (atlased) content has less to gain
- Does not suffer from atlasing restrictions
  - Can still tile textures
  - All textures don't need to be squeezed onto a single sheet

Niche product

- ~10 sales a month
- I have not done any marketing
- Very useful in contracting work
  - Draw calls a major issue for people asking me to optimize their games



A review of Anti-Tiling techniques



- Tiling textures a major issue in graphics
- Especially problematic on terrains
  - Best selling module for MicroSplat is the Anti-Tiling module
- Lots of attempts to cover it up over the years

Classic Technique: Distance Resampling

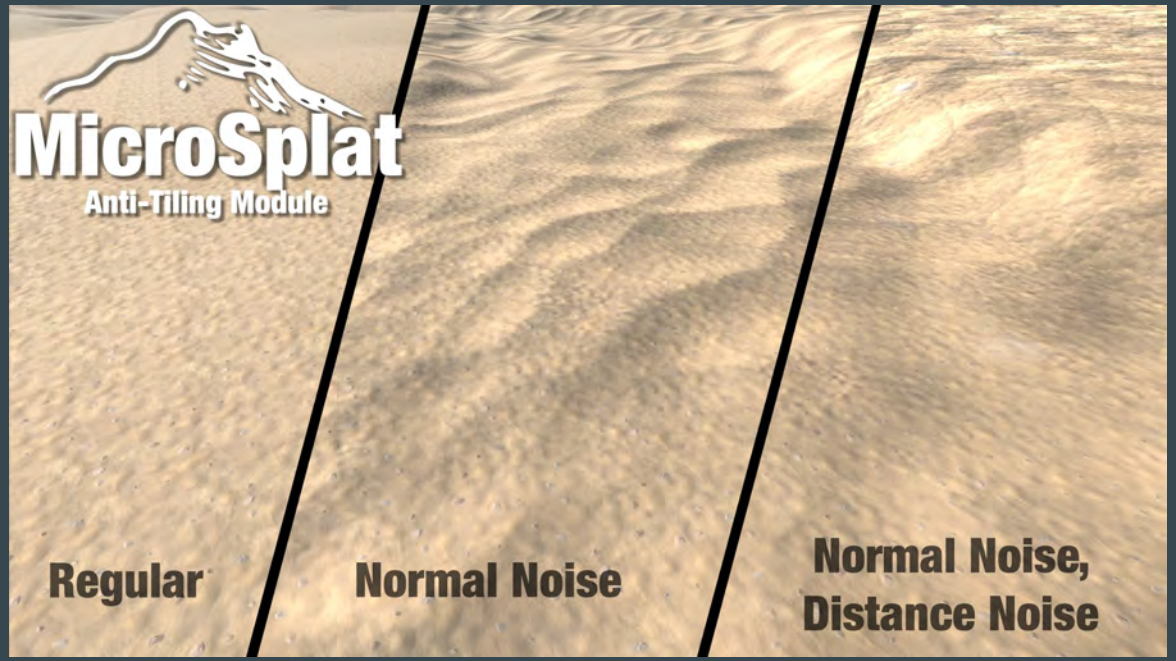
- AKA: UV mixing, etc..
- Basic idea:
  - Sample textures
  - Sample textures again at lower UV tiling rate
  - Blend result together based on distance from camera
- Looks ok, but
  - Blend is obvious
  - Doubles samples used
- Improvements (in MicroSplat)
  - Only sample diffuse layer
  - Height blend layers
  - Only sample top 2 weighted textures

Classic Technique: Macro/Micro texturing

- Basic idea:
  - Tile a small or large scale texture inside of the main one
  - First used in Unreal (the game)
  - Texture usually greyscale, and uses overlay or blend2x operator
- Fast and cheap
  - But doesn't fix tiling, rather creates large and small scale details to increase effective resolution
- Available in MicroSplat AntiTile

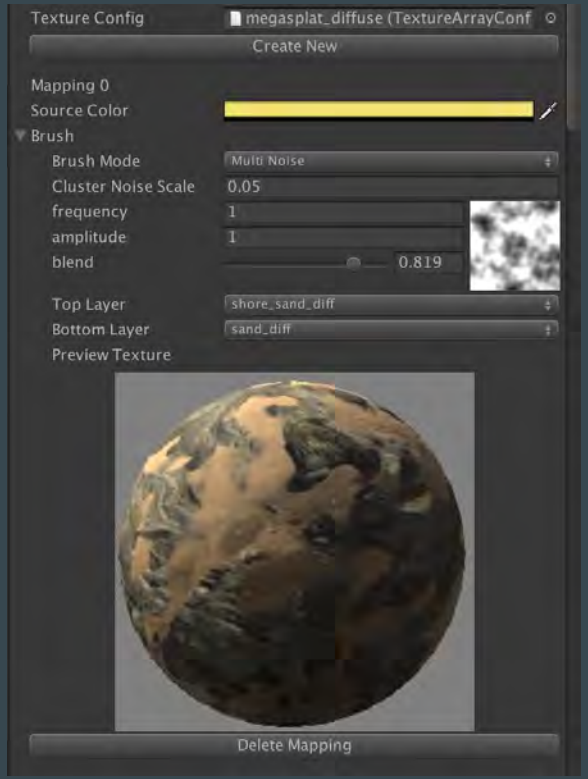
Classic Technique: Noise Normal

- Basic idea:
  - Blend a normal map in at a larger scale to create lighting variations across the surface
- Cheap, fast, works well
- Can have all of these per texture in MicroSplat



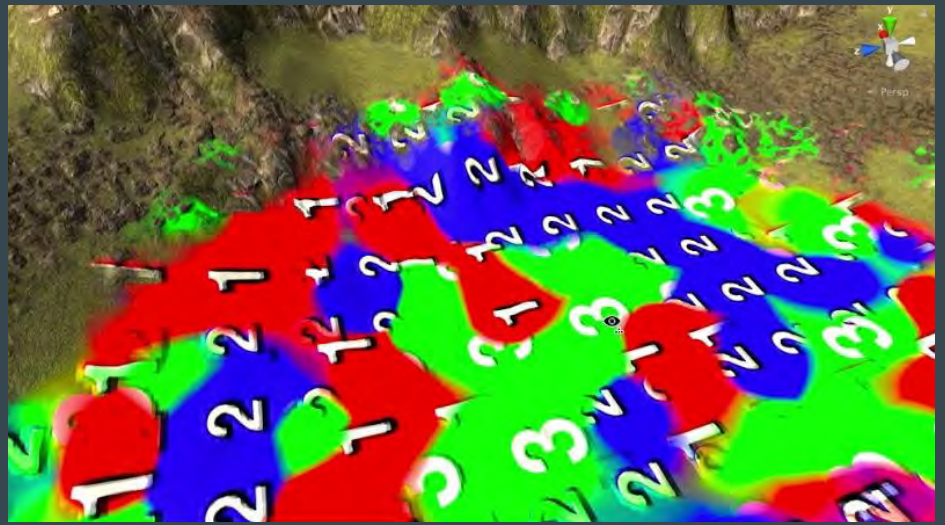
New Technique: Texture Clustering

- First introduced in MegaSplat
  - Intrinsic to how it blends textures
  - Answers my One Question : “What can you do with hundreds of textures?”
- Idea
  - Since MegaSplat has to sample three texture sets per face (one for each vertex), why not vary which texture gets sampled?
- First class system in MegaSplat
  - Cluster concept worked through all tools
  - Cluster can contain any number of textures, chosen by noise, slope, or height ranges

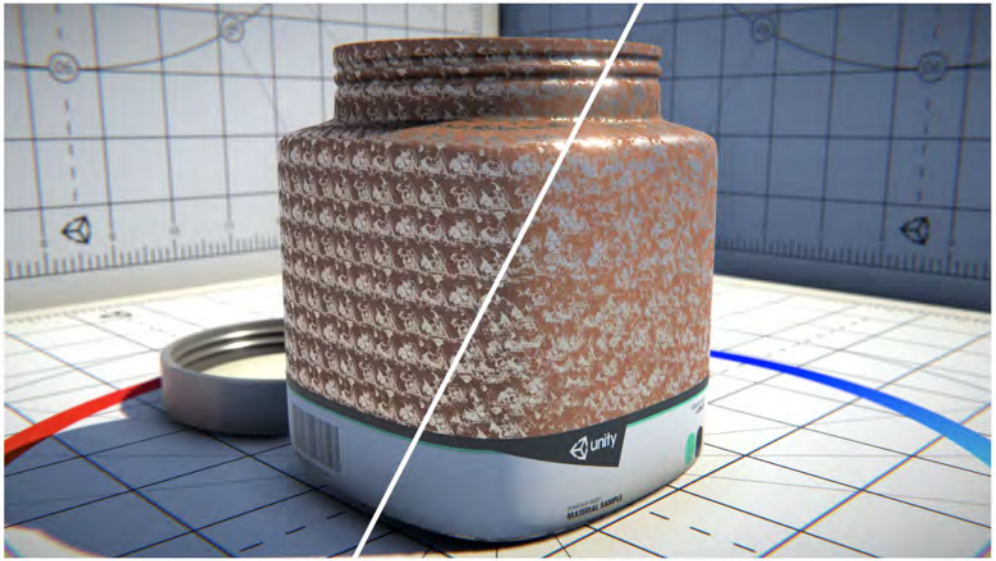


MicroSplat Texture Clustering

- Cannot use MegaSplat technique
  - Uses traditional splat weights
- Solution:
  - Sample a noise texture
  - Height blend between 3 textures
  - Store in parallel texture arrays
- 3x sample count, so expensive compared to MegaSplat
  - But easier to control
  - Not tied to vertex/control texture resolution
  - Not baked into paint job



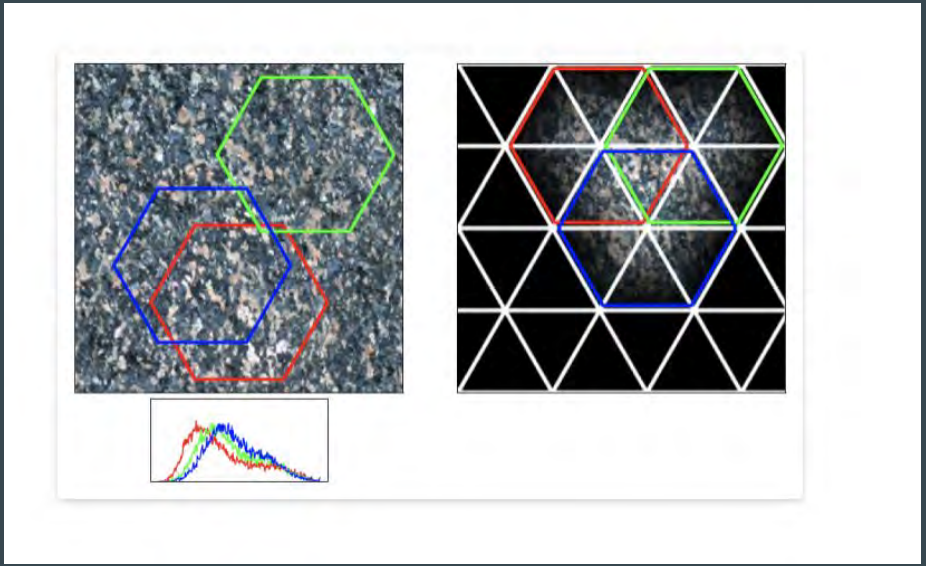
Stochastic Texture Sampling



Procedural Stochastic Texturing

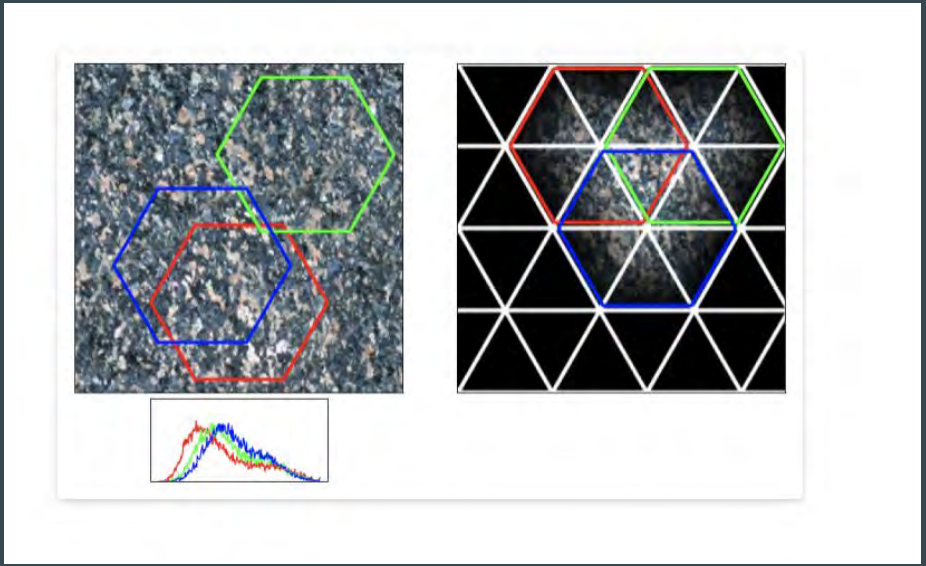
- Based on a paper by Eric Heintz
- Unity Labs released a version in February
- Anti-tiling technique

Basic Idea



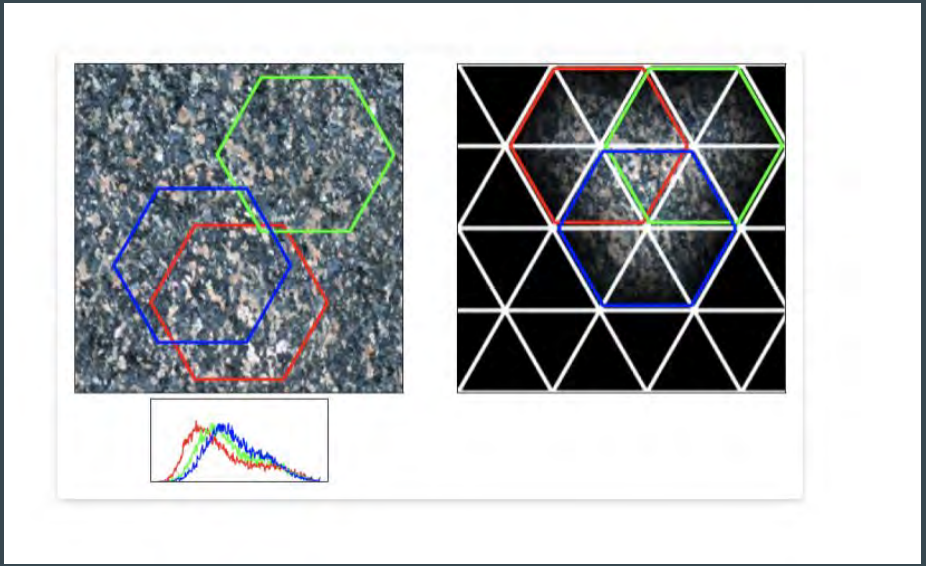
- Construct 3 virtual triangles, offset and randomized
- Sample textures 3 times
- Blend preserving histogram of original image

Massive Complexity for histogram preserving blend



- Must preprocess textures into custom format (SLOW)
- Must generate lookup tables and store in additional textures
- Need custom code based on texture format
  - Only uncompressed and DXT1 supported
- Lots of math, lots of code, dependent texture reads

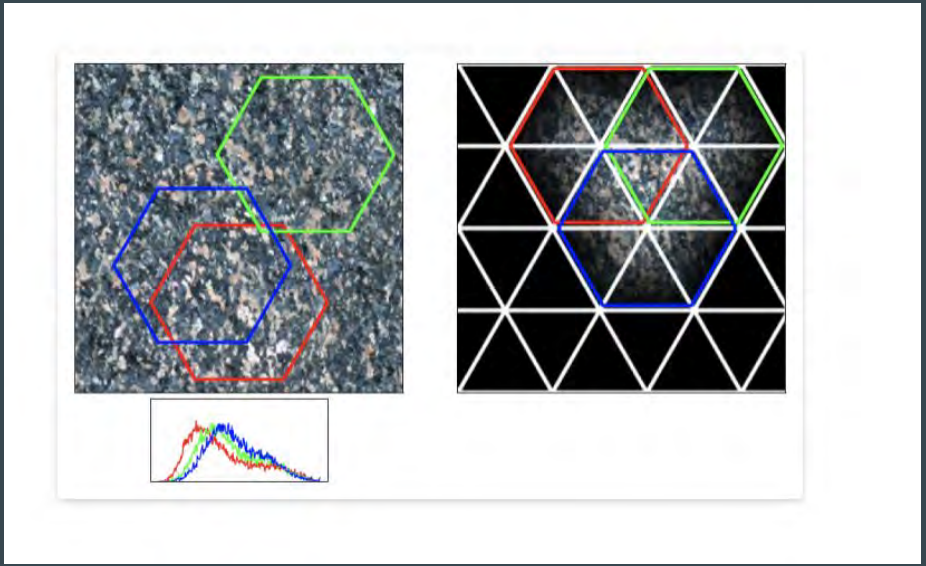
Basic Idea



- Construct 3 virtual triangles, offset and randomized
- Sample textures 3 times
- Blend preserving histogram of original image

This sounds familiar

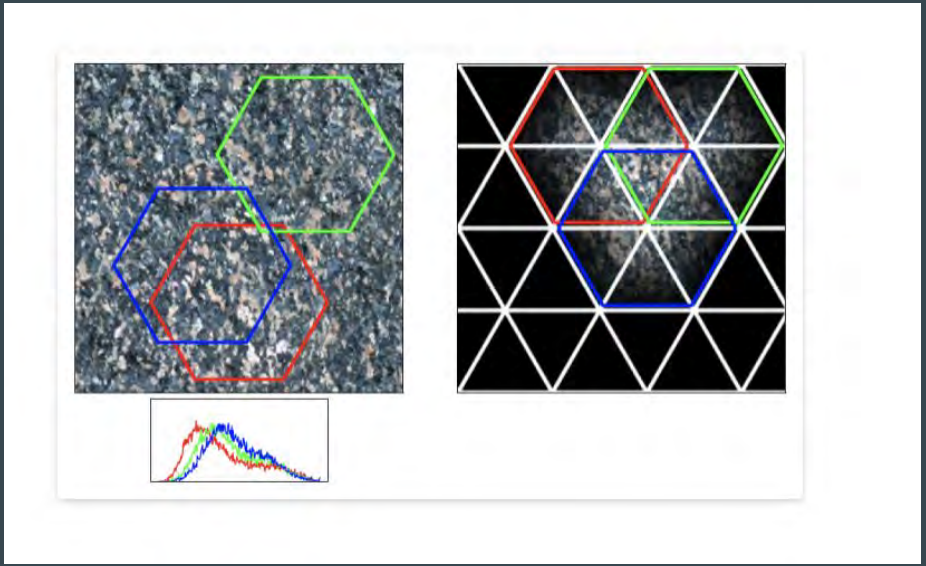
Basic Idea



- Construct 3 virtual triangles, offset and randomized
- Sample textures 3 times
- Blend preserving histogram of original image

This sounds familiar

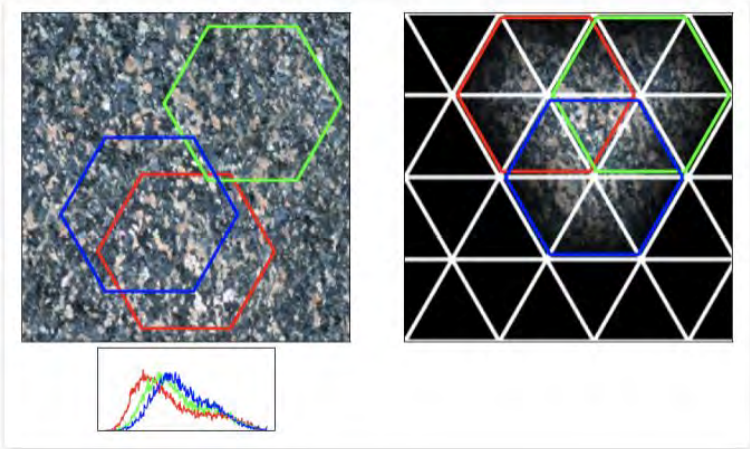
Basic Idea



- Construct 3 virtual triangles, offset and randomized
- Sample textures 3 times
- Blend preserving histogram of original image

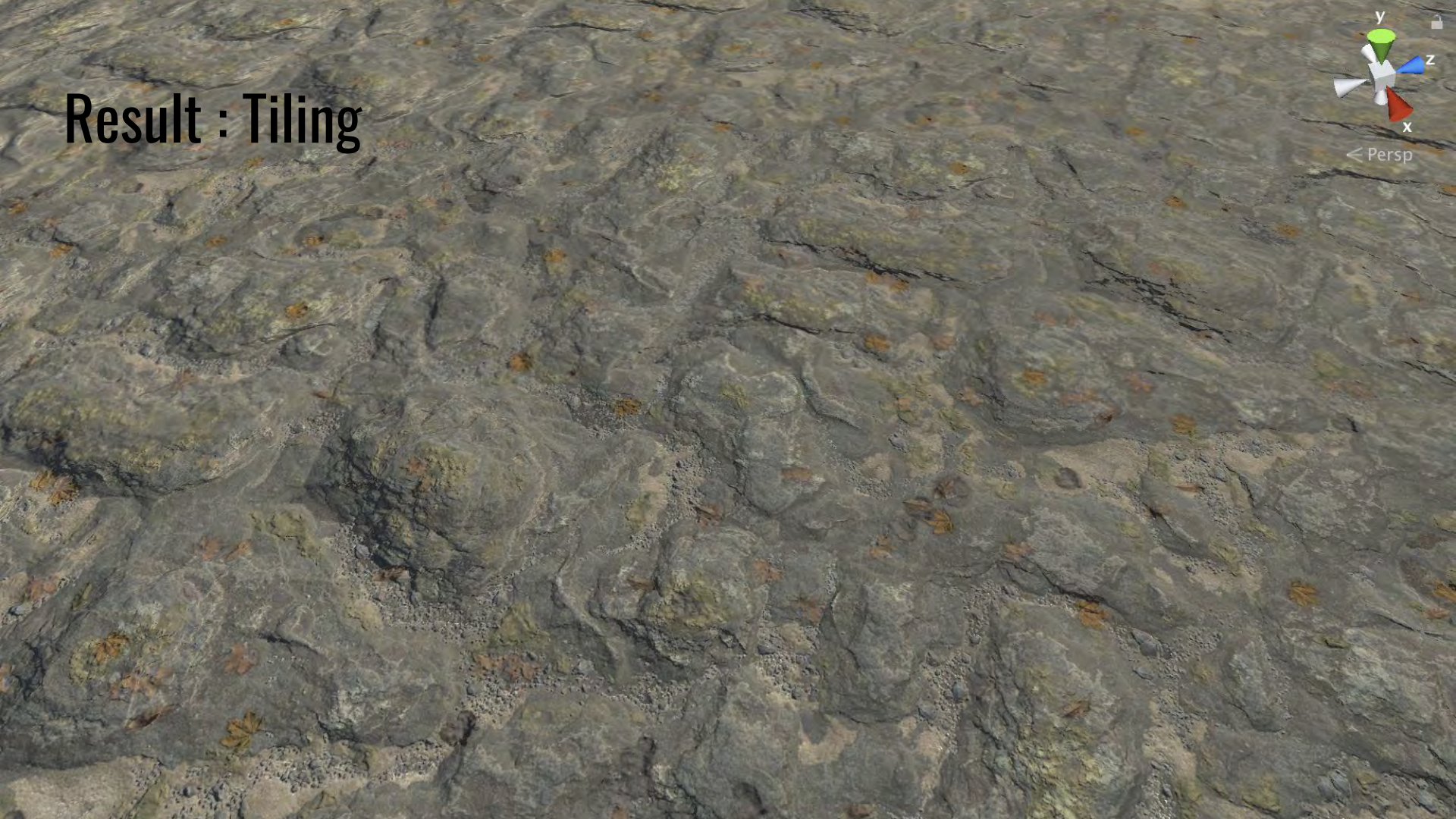
This sounds overly complex

Basic Idea

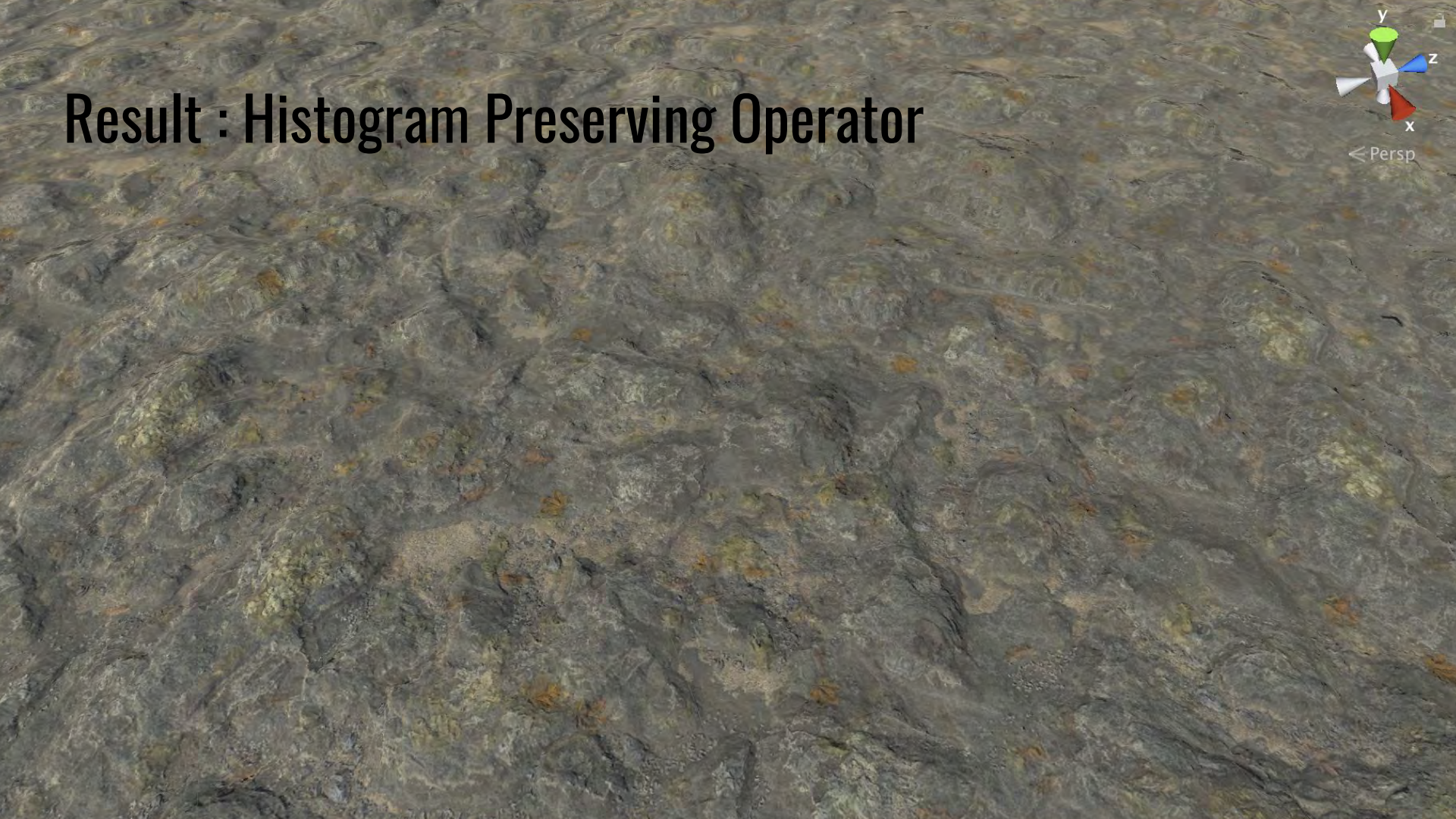


- Construct 3 virtual triangles, offset and randomized
- Sample textures 3 times
- ~~Blend preserving histogram of original image~~
- **Blend using height map operator**
  - Could use luminosity or other term if height map is not available

Result : Tiling



Result : Histogram Preserving Operator



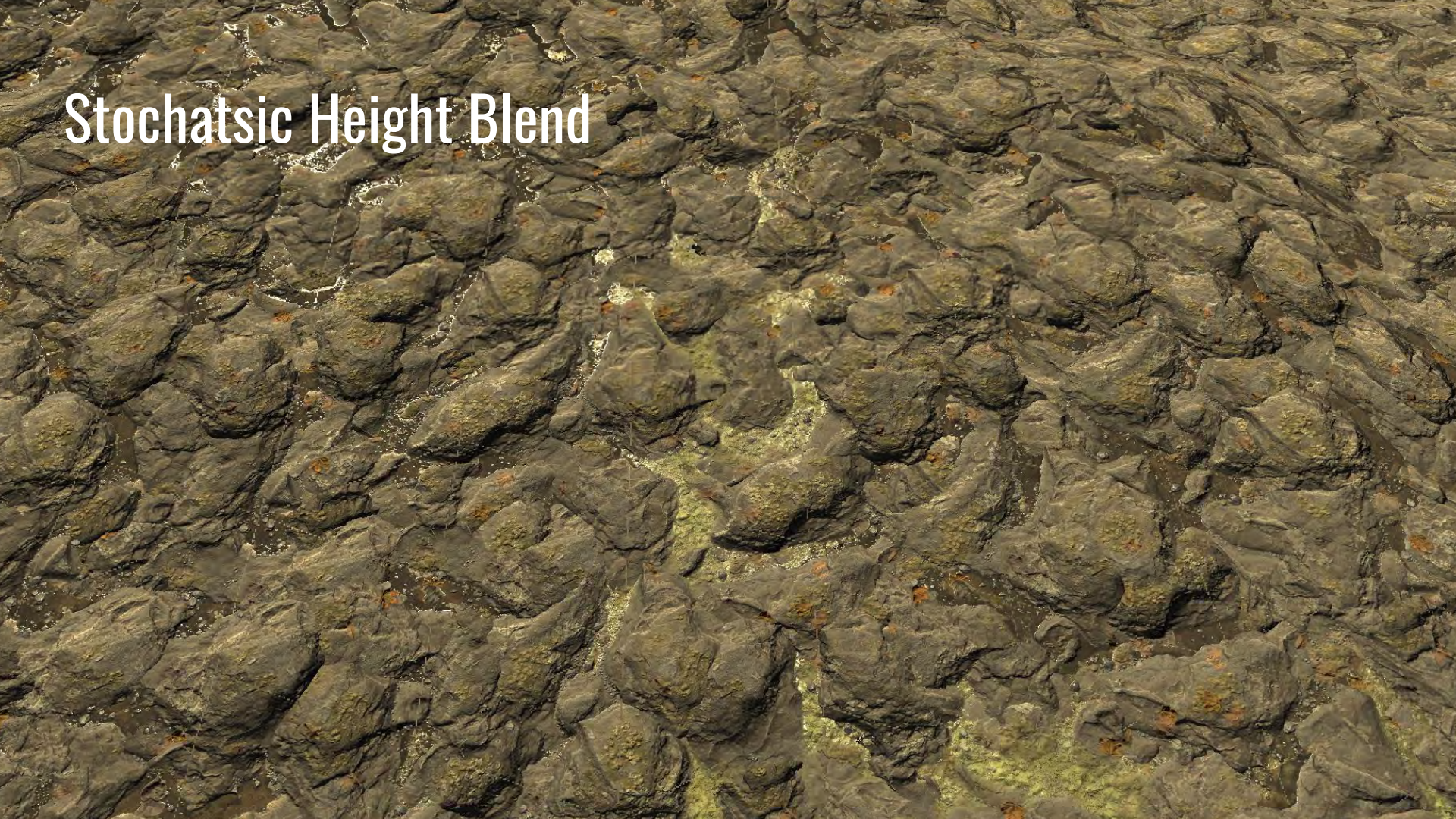
Result : Height Blend Operator

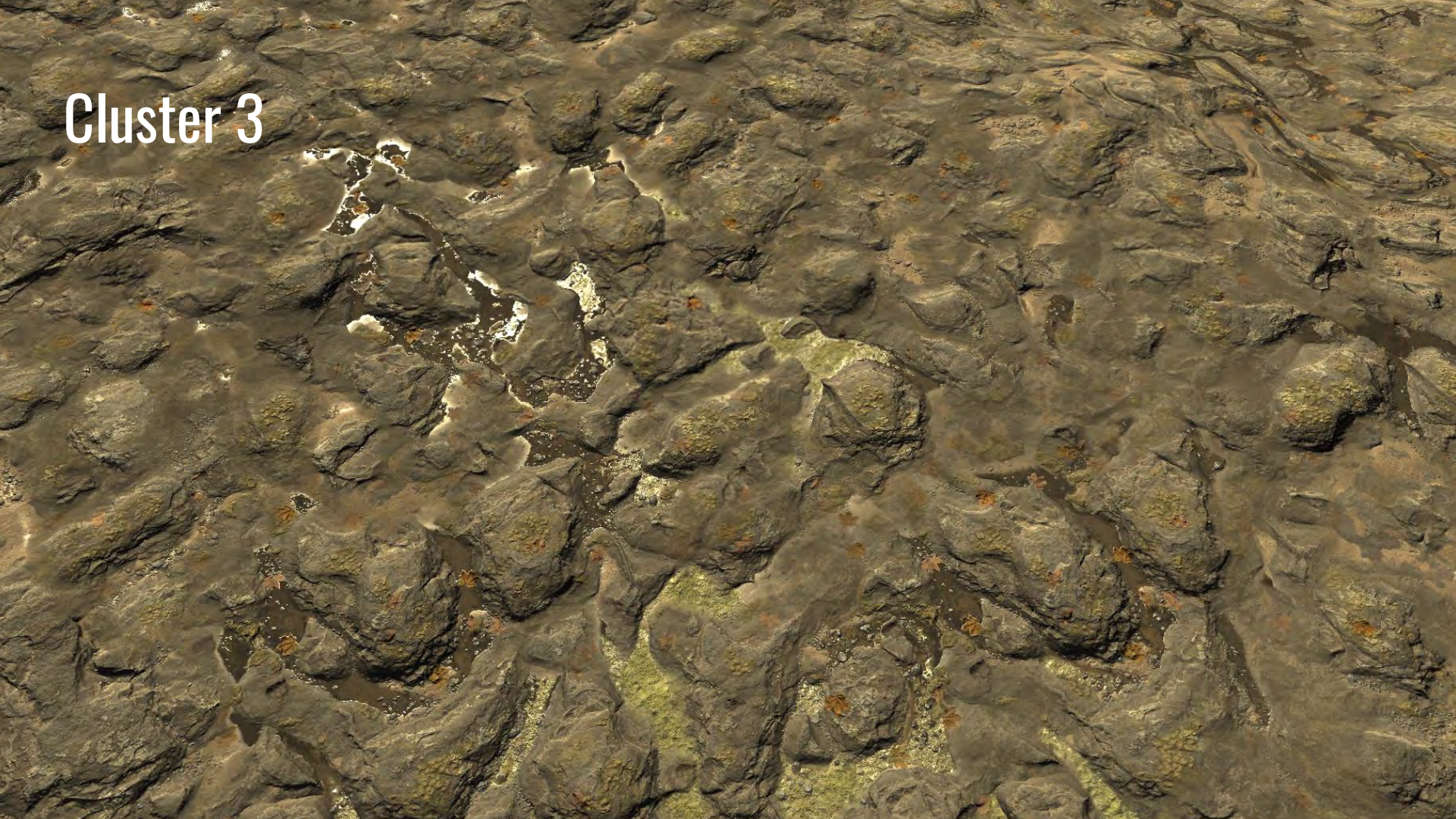
Advantages

- No preprocessing, no LUTs, works with any texture format, less code
- Retains look of original texture better
  - Height map operator has adjustable blend width
    - This means you see more of your texture un-modified
  - Histogram Preserving operator blends across entire virtual triangle
    - Everywhere is blending
    - Everything smearing towards sameness
- Added adjustable triangle scale parameter for user
  - Larger triangles show more of texture intact (but might exhibit tiling)
  - Smaller triangles synthesize more

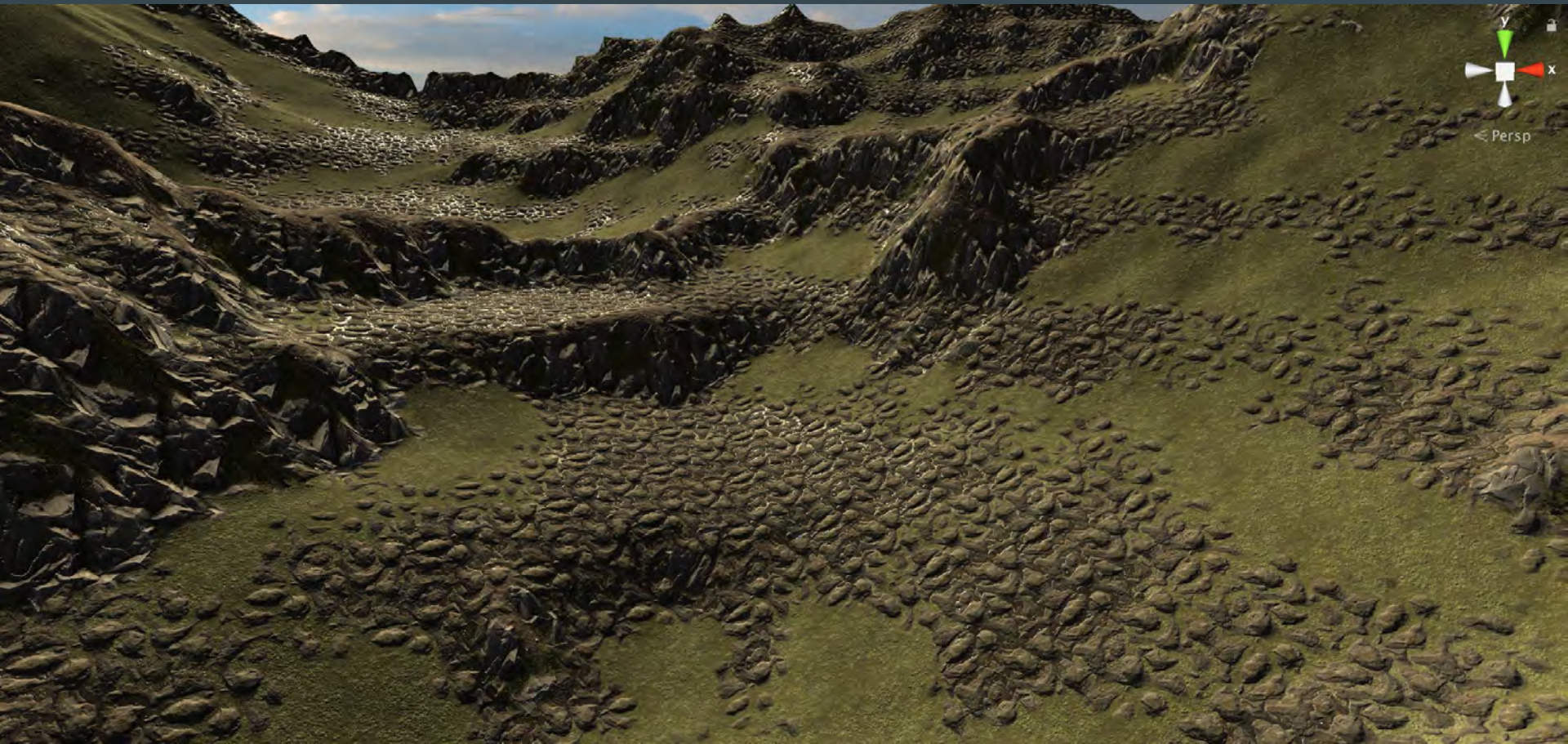
Tiling

Stochatsic Height Blend



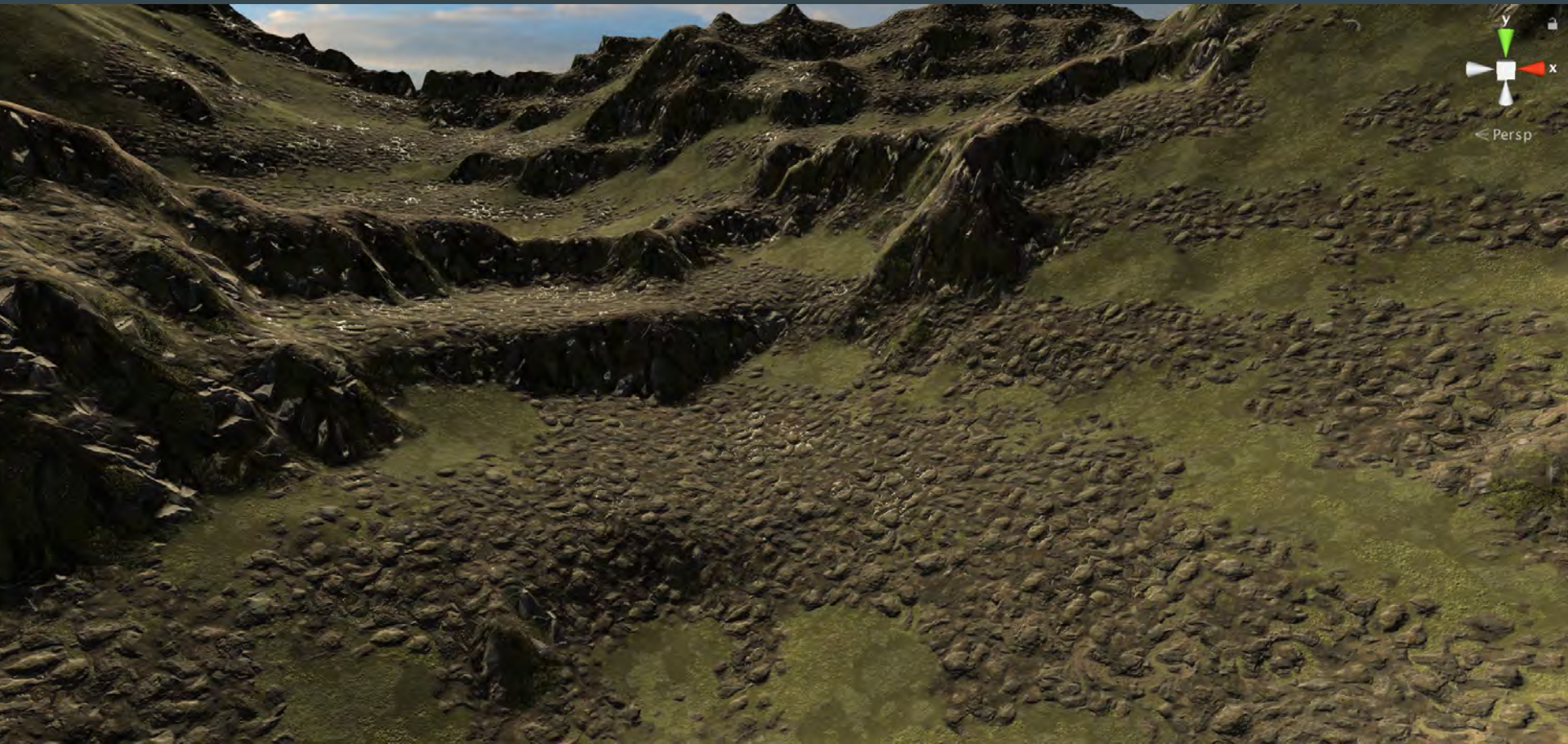

Cluster 3

Tiled



Stochastic Height Blend

Cluster 3



Research area

Histogram preserving operator:

- Biases too much towards a wash of the texture
- Frequency of features is increased

Height map operator

- Biases towards peaks of height map
  - Could bias to preserve valleys?
- Frequency of features is increased
- Luminosity of albedo often works surprisingly well

Overall

Height Map operator is superior to Histogram preserving blend

- No preprocessing
- No extra textures or custom texture formats
- Simpler shader code
- No extra uniforms needed
- No dependencies on compression format
- Preserves detail of original texture better
- Blend areas minimized, overall higher quality
- Other data can be used when height map is not available
    - Luminosity, Height from Normal, noise, etc..

Current Techniques in MicroSplat

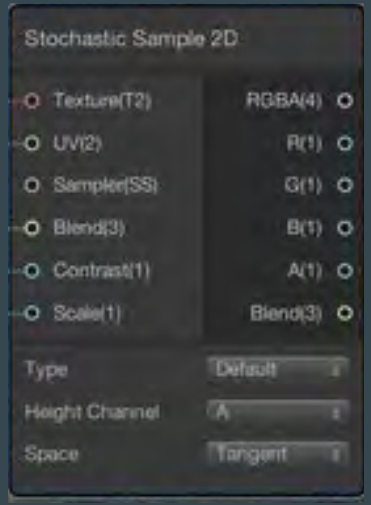
- Anti-tile
  - Distance Resampling
  - Detail Noise
  - Distance Noise
  - Normal Noise
- Texture Clusters
  - Stochastic Height Blend
  - Texture Cluster 2 layer
  - Texture Cluster 3 layer

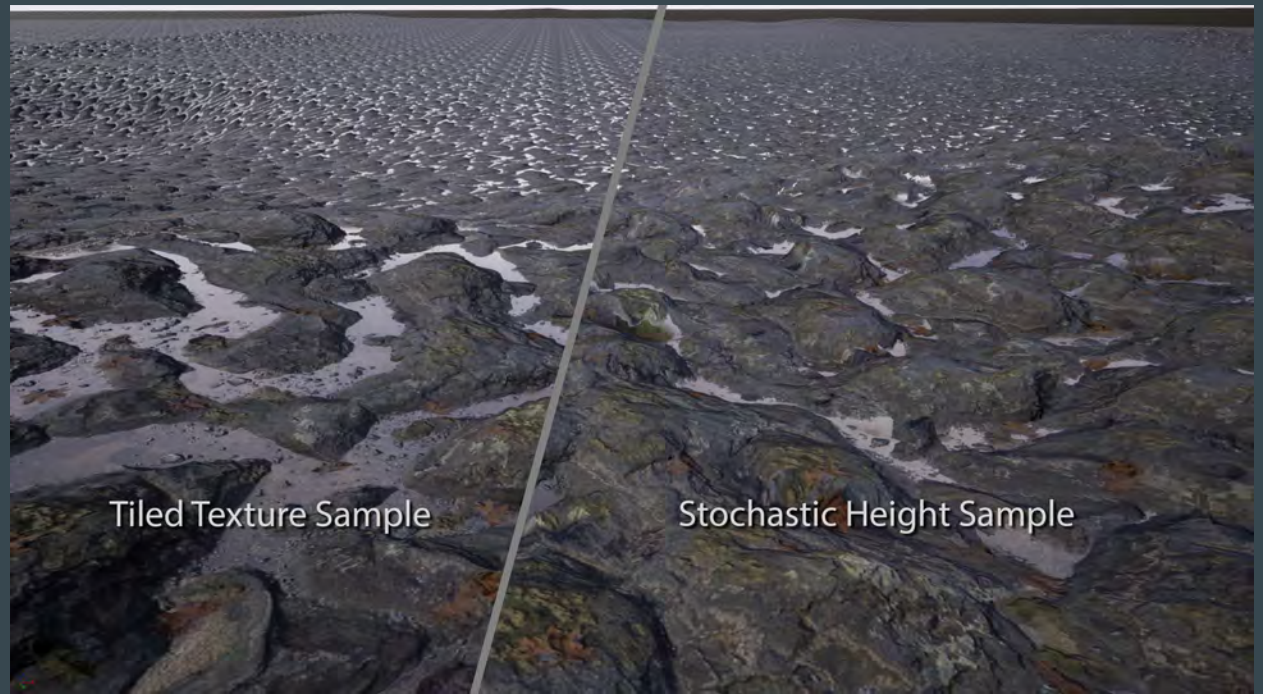
One Click

Require authoring texture variations

Stochastic Height Blend Node

- For Unity Shader Graph, Amplify Shader Editor, UE4 shader graph
- Blends based on selectable texture channel or luminosity of RGB channels





Shader Compilers

- Incredibly good at code stripping
  - Structs are not real, so will even strip unused structure members
    - Use structs to organize data
  - Unity's compiler sometimes strips things you're actually using though
    - o.Albedo *= saturate(i.viewDir + 2);
  - Lots of shaders (even Unity's) do tons of manual optimizations which their compiler would strip
    - Makes code more complex than it needs to be
- Can identify repeated texture samples and combine them
- Reorders code to start texture fetches as soon as possible
- Check generated and compiled output


Questions?
slipster216@gmail.com
@slipster216