Terrain Rendering in Frostbite using Procedural Shader Splatting

Johan Andersson
Rendering Architect, EA DICE

SIGGRAPH2007

FROSTBITE
A DICE TECHNOLOGY

Outline

- Previous games
- Terrain overview
- Graph-based shaders
- Terrain shading & texturing
- Terrain rendering
- Undergrowth
- Conclusions
- Future

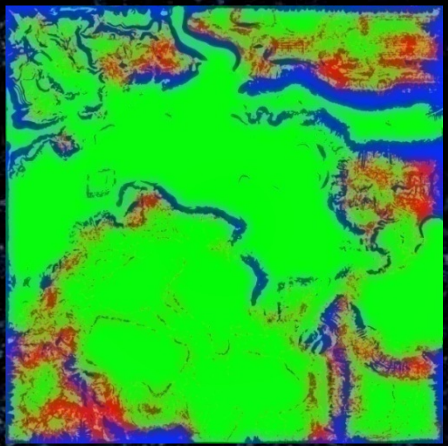
Outline

- Previous games
- Terrain overview
- Graph-based shaders
- Terrain shading & texturing
- Terrain rendering
- Undergrowth
- Conclusions
- Future

Battlefield 2 terrain

- ”Traditional terrain rendering”
- Static geometry
  - Heightfields → optimized meshes
- Unique low-res color map
- Tiled detail maps
  - 3-6 detail maps
  - Controlled by unique mask textures
  - Macro detail map
- Fixed shading & compositing
  - Expensive & difficult to add features
  - Special case for mountains/slopes
- No destruction ☹

Color map



Detail mask map

Our requirements

- Battlefield: Bad Company
  - Frostbite engine pilot project
- Xbox 360 & PS3
- Low memory usage
  - Scaling up BF2 methods (~45mb) not possible
- High detail up close and far away
  - Long view distance (16 km)
  - Normal maps, multiple texturing techniques
- Destruction
  - Affecting geometry, texturing, shading

Terrain overview

- Multiple high-res heightfield GPU textures
  - Easy destruction
  - Used in both VS and PS
  - 16-bit unsigned integer format (L16)
- Normals calculated in the shader from heightfield
  - Very high detail lighting in a distance
  - Saves memory

Terrain lighting screenshot

DICE SIGGRAPH2007 FROSTBITE A DICE TECHNOLOGY

Terrain texturing - general idea

- Compute instead of store
  - Shading, texturing, material compositing
  - Using procedural techniques in shaders
  - Allows changing & adding materials dynamically
    - Destruction!
- Splat arbitrary shaders over the terrain
  - Artist created
  - Determines both look and visibility of materials
  - Specialized to requirements of material
  - *"Procedural Shader Splatting"*

Terrain material requirements

- Different shading & texturing requirements depending on
  - Natural complexity
  - Distance from camera
  - Importance in game
  - How well used it is (effort to create)
  - And more
- Example cases
  - Seafloor material (partially obscured)
  - Parallax mapping only on rocky surfaces

Specialized terrain material shaders

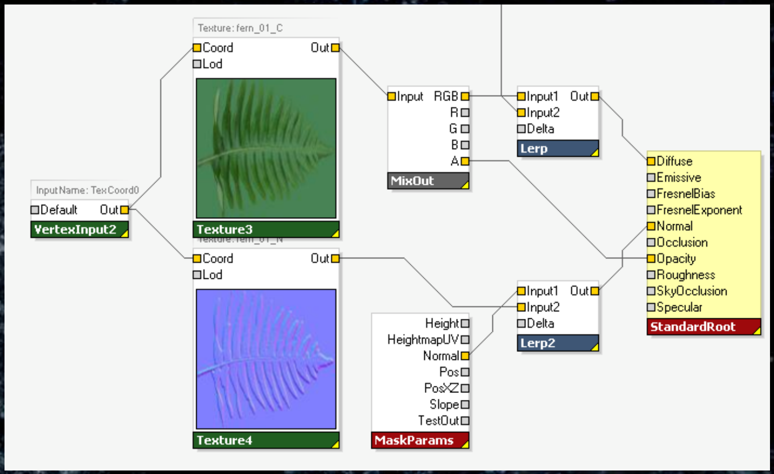
- Vary features & complexity
  - Texture compositing, side-projection, normalmapping, parallax-mapping, specular
- Flexible tradeoff of memory, performance and quality for each material
  - Can be complex for artists
  - How we've always done shaders for other types of geometry

Outline

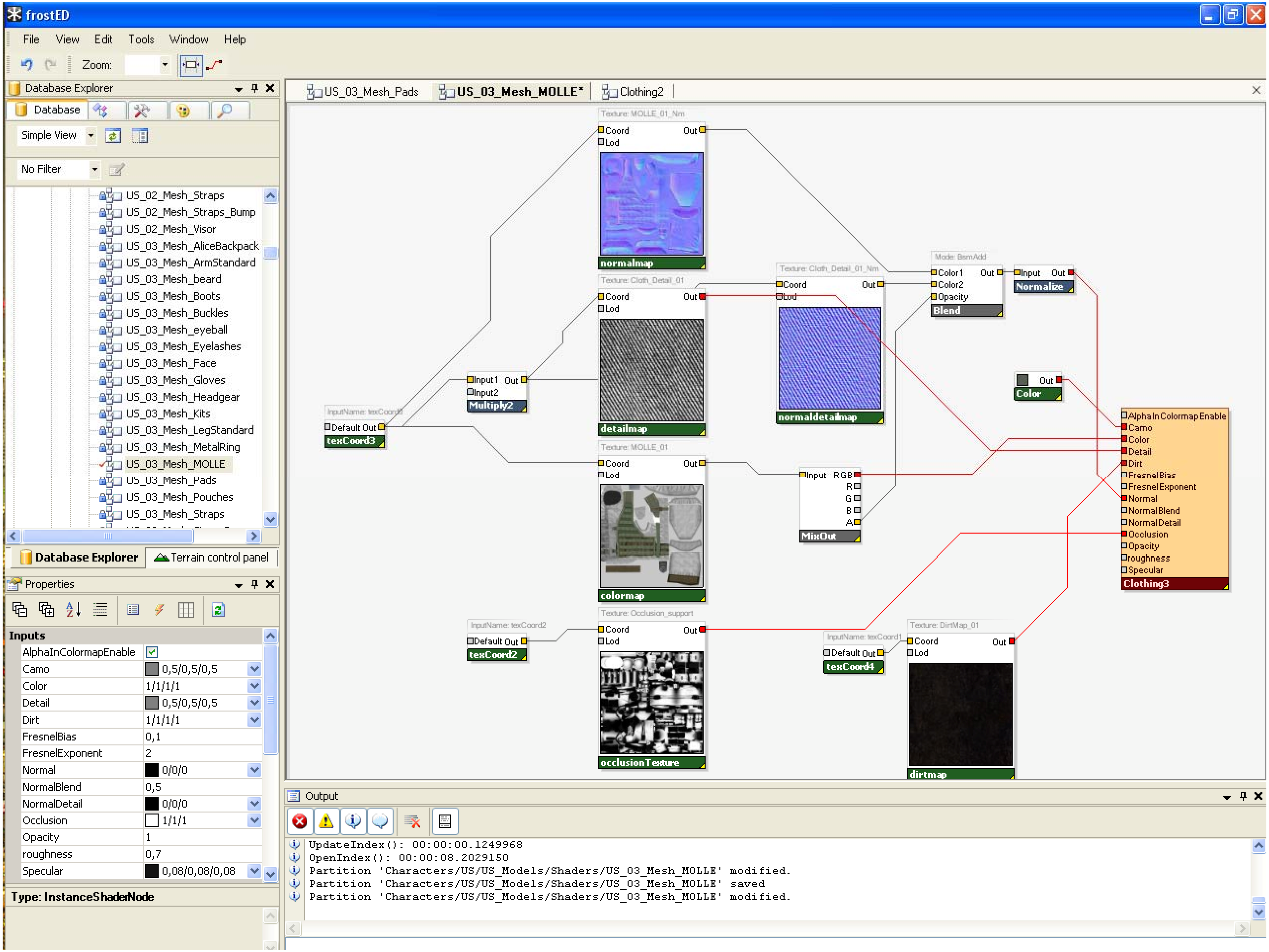
- Previous games
- Terrain overview
- Graph-based shaders
- Terrain shading & texturing
- Terrain rendering
- Undergrowth
- Conclusions
- Future

Graph-based surface shaders

- Rich high-level shader framework
  - Used by all meshes & systems incl. terrain
- Artist-friendly
  - Easy to create, tweak and manage
- Flexible
  - Programmers & artists can extend & expose features
- Data-centric
  - Encapsulates resources
  - Can create or transform shaders in automated processes

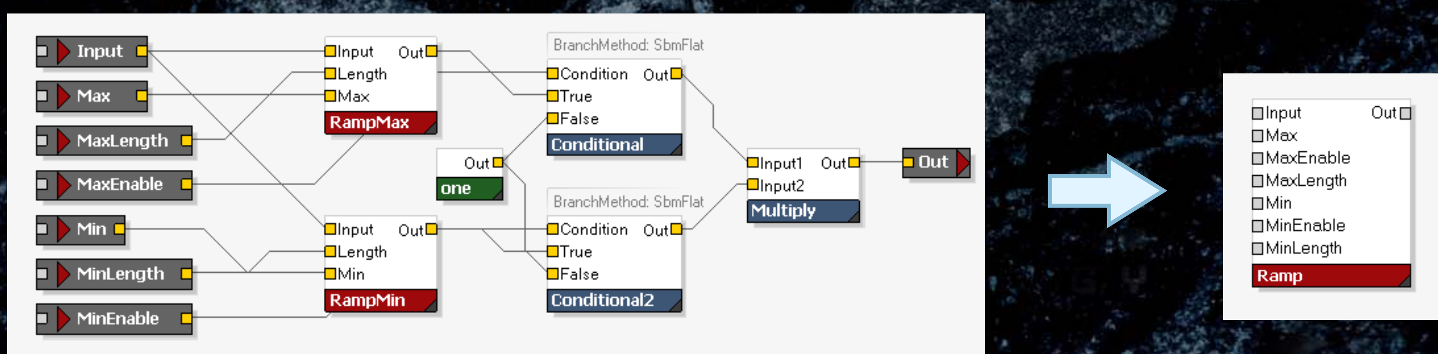


Example surface shader graph


frostED
File View Edit Tools Window Help
Zoom:
Database Explorer
Database
Simple View
No Filter
US_02_Mesh_Straps
US_02_Mesh_Straps_Bump
US_02_Mesh_Visor
US_03_Mesh_AliceBackpack
US_03_Mesh_ArmStandard
US_03_Mesh_beard
US_03_Mesh_Boots
US_03_Mesh_Buckles
US_03_Mesh_eyeball
US_03_Mesh_Eyelashes
US_03_Mesh_Face
US_03_Mesh_Gloves
US_03_Mesh_Headgear
US_03_Mesh_Kits
US_03_Mesh_LegStandard
US_03_Mesh_MetalRing
US_03_Mesh_MOLLE
US_03_Mesh_Pads
US_03_Mesh_Pouches
US_03_Mesh_Straps
Database Explorer
Terrain control panel
Properties
Inputs
AlphaInColormapEnable
Camo 0,5/0,5/0,5
Color 1/1/1/1
Detail 0,5/0,5/0,5
Dirt 1/1/1/1
FresnelBias 0,1
FresnelExponent 2
Normal 0/0/0
NormalBlend 0,5
NormalDetail 0/0/0
Occlusion 1/1/1
Opacity 1
roughness 0,7
Specular 0,08/0,08/0,08
Type: InstanceShaderNode
US_03_Mesh_Pads
US_03_Mesh_MOLLE*
Clothing2
Texture: MOLLE_01_Nm
Coord Out
Lod
normalmap
Texture: Cloth_Detail_01
detailmap
Texture: Cloth_Detail_01_Nm
normaldetailmap
Mode: BsmAdd
Color1 Out
Color2
Opacity
Blend
Input Out
Normalize
Out
Color
Input1 Out
Input2
Multiply2
InputName: texCoord1
Default Out
texCoord3
Texture: MOLLE_01
colormap
Input RGB
R
G
B
A
MixOut
AlphaInColormapEnable
Camo
Color
Detail
Dirt
FresnelBias
FresnelExponent
Normal
NormalBlend
NormalDetail
Occlusion
Opacity
roughness
Specular
Clothing3
InputName: texCoord2
texCoord2
Texture: Occlusion_support
occlusionTexture
InputName: texCoord1
texCoord4
Texture: DirtMap_01
dirtmap
Output
UpdateIndex(): 00:00:00.1249968
OpenIndex(): 00:00:08.2029150
Partition 'Characters/US/US_Models/Shaders/US_03_Mesh_MOLLE' modified.
Partition 'Characters/US/US_Models/Shaders/US_03_Mesh_MOLLE' saved
Partition 'Characters/US/US_Models/Shaders/US_03_Mesh_MOLLE' modified.

Instance shaders

- Shader graph network that can be instanced as a node in another shader
  - Think C/C++ functions
- Reduces complexity and allows reuse
- Hide and encapsulate functionality on multiple levels
  - Choose inputs & outputs to expose



Shader pipeline

- Big complex offline pre-processing system
  - Used surface shaders & states is gathered per level
- Generates shading solutions
  - HLSL vertex and pixel shaders
  - States, constants, passes
- Can trade shader efficiency for amount of shader permutations
  - Example: include fog in all shaders or create seperate shaders with and without fog
  - Lots of optimization opportunities

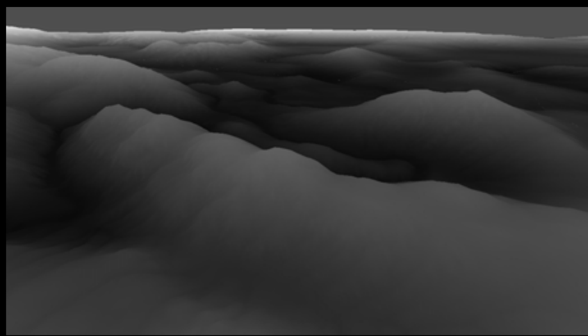
Outline

- Previous games
- Terrain overview
- Graph-based shaders
- Terrain shading & texturing
- Terrain rendering
- Undergrowth
- Conclusions
- Future

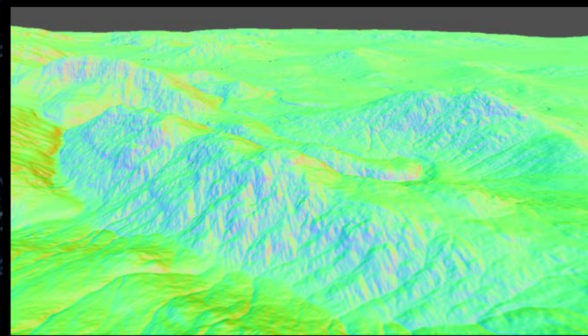
Procedural techniques

- There are many interesting procedural texturing techniques
  - For example: Wang tiles
- But most are
  - Heavy or difficult to run on a GPU
  - Difficult to mipmap correctly
  - Require rendering to offscreen targets
- Want direct evaluation techniques that can be executed directly inside shaders
  - At least as a base
  - Only computes visible pixels

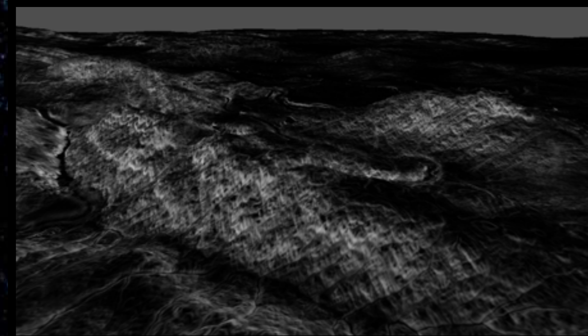
Procedural parameters



Height



Normal



Slope

- Build procedural patterns of basic terrain parameters evaluated in all shaders
  - Uses the GPU heightfield textures
- Commonly used in offline terrain rendering and texture generation
  - Such as Terragen

Normal filtering

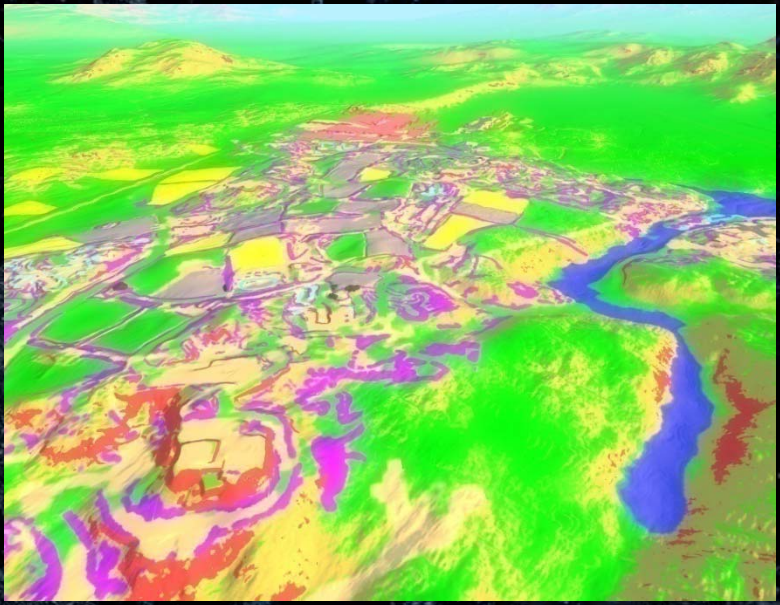
- Simple & fast cross filter
  - Not correct at diagonals, but good enough for us
- Result is world space normal

```
float3 filterNormal(float2 uv, float texelSize, float texelAspect)
{
    float4 h;
    h[0] = hmap.Sample(bilSampler, uv + texelSize*float2( 0,-1)).r;
    h[1] = hmap.Sample(bilSampler, uv + texelSize*float2(-1, 0)).r;
    h[2] = hmap.Sample(bilSampler, uv + texelSize*float2( 1, 0)).r;
    h[3] = hmap.Sample(bilSampler, uv + texelSize*float2( 0, 1)).r;

    float3 n;
    n.z = (h[0] - h[3]) * texelAspect;
    n.x = (h[1] - h[2]) * texelAspect;
    n.y = 2;
    return normalize(n);
}
```

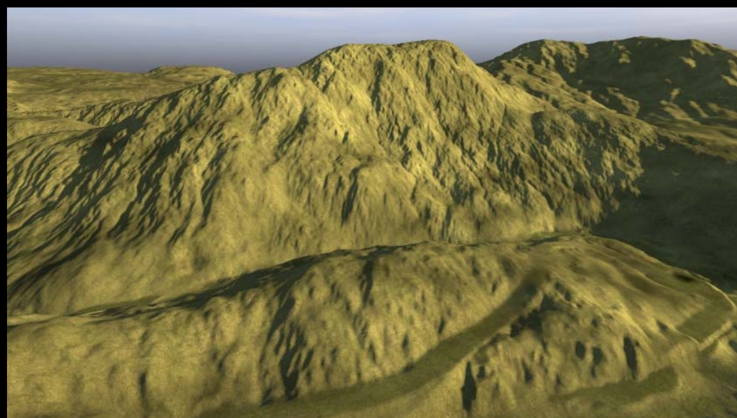
Material masking

- Terrain shaders determine material mask
  - I.e. visibility
- Can use
  - Procedural parameters
    - Typically slope
  - Painted masks
  - Arbitrary texturing or shader computation
  - Or combine them all

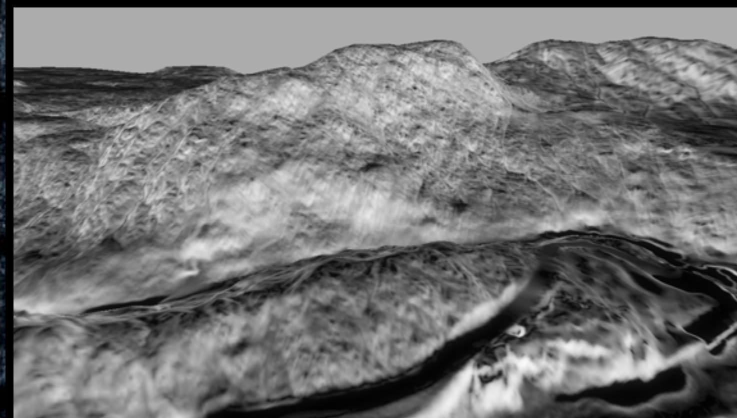


Multiple material masks of all types
Red = slope-based cliff material
Pink = painted dirt material

Mountain material example

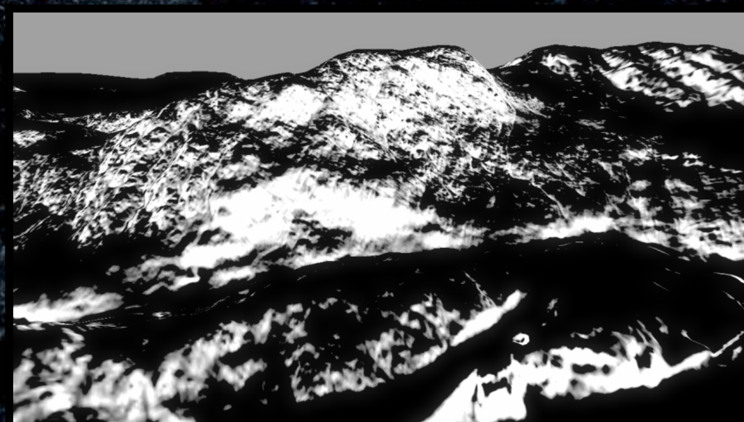


Only grass material



Slope

With mountain material



Mask (slope scaled & biased)

Painted masks

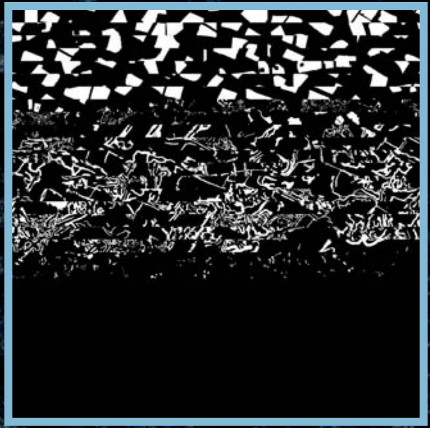
- Many materials can not be solely distributed on a procedural basis
  - Fields, man-made areas, artist control
- Support painted per-material masks
  - Memory heavy but flexible
  - 0.5 – 8 pixels/meter
- Coverage typically low
  - 5-15% coverage of levels
  - Not much overlap

Fields with painted masks

Static sparse mask textures (1/2)

- Store painted masks in sparse quadtree textures
  - Major memory reduction
- Split painted masks into 32x32 tiles and store in atlas texture
  - Only unique tiles
  - DXT5A compression
  - Can use texture arrays
    - But want more than 64/512 slices

Source mask



Atlas texture

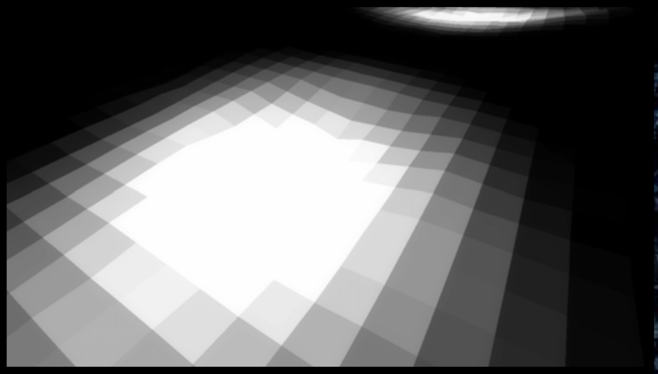
Static sparse mask textures (2/2)

- Tile index & level textures cover the terrain
  - Tile index: 16-bit integer index into tile atlas
  - Tile level: 8-bit integer. Size of tile world area
  - Low-res, 16 meters/pixel
- Lookup with world-space position
- Calculate atlas texture coordinates
  - Details in course notes
- 4 masks packed together in RGBA
  - For efficiency and to reduce # of samplers

Destruction mask (1/2)

- Change material and/or look around craters
- Render decals into destruction mask texture
  - Covers playable area (2x2 or 4x4 km)
  - Usually 2 pixels/meter
- Material shaders get access to simple 0-1 value
  - Blends in or replaces textures and colors

Crater screenshot

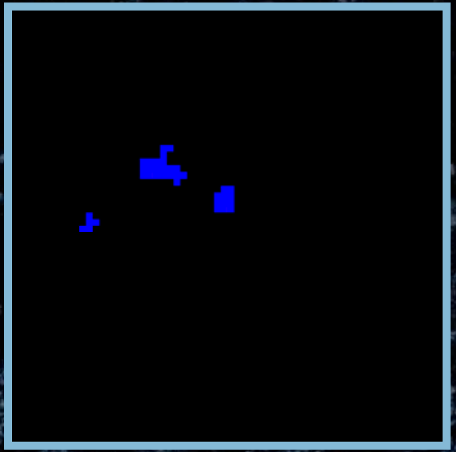


Crater mask (point-filtering)

Destruction mask (2/2)

- Observation: 100% destruction not possible in practice
  - Due to gameplay
- Can store mask as sparse texture to save memory
  - Indirection texture covers whole area
    - RG88, 128x128 resolution = 16m cells
  - .rg indexes into atlas texture
    - 64x64 L8 tiles
- Gives virtual 8192x8192 texture with tweakable max coverage
  - 16.7 mb -> ~1.7 mb (10% coverage)

Atlas texture



Indirection texture

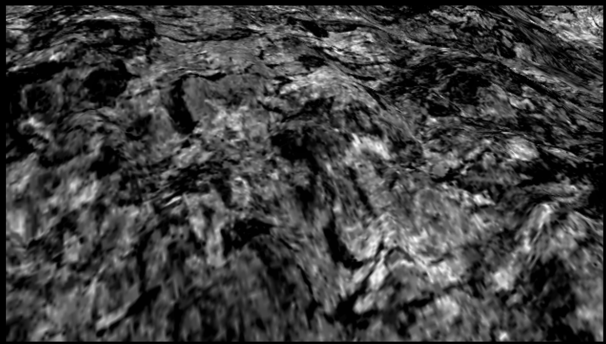
Increasing mask detail

- All the masking techniques can suffer from bluriness due to low resolution
  - Add detail in the shaders!
- Many methods for generating detail
  - fBm, noise
  - Detail textures
  - Reusing textures with scale, bias and contrast
    - Colormaps, normalmap.b (”occlusion”)
- And for compositing/blending in the detail
  - Multiply, add, min, max, overlay, custom
  - Fully programmable since in shaders

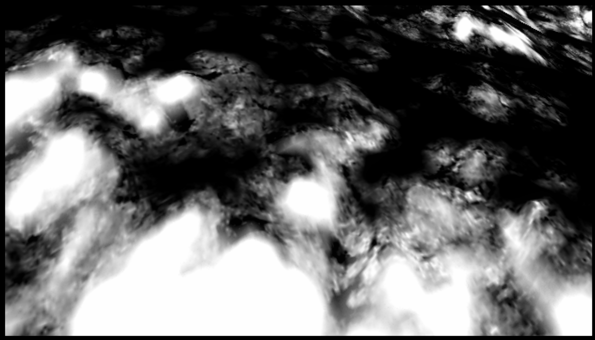
Photoshop Overlay blend

- Perfect for blending in high-frequency details
- Doesn't affect areas where base mask is 0.0 or 1.0
  - Good for dynamic flow control

Base mask from slope



Detail mask

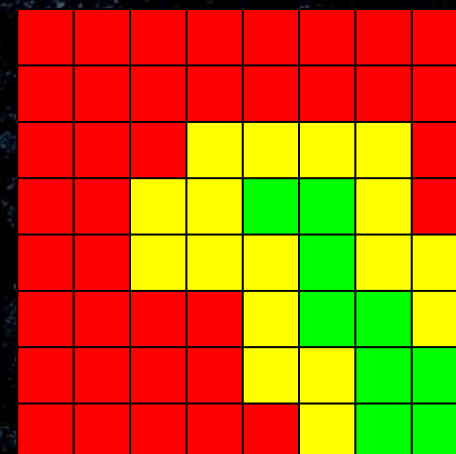


Overlay blend result

```
float overlayBlend(float base, float value, float opacity)
{
    float a = base < 0.5 ? 2*base*value : 1 - 2*(1-base)*(1-value);
    return lerp(base, a, opacity);
}
```

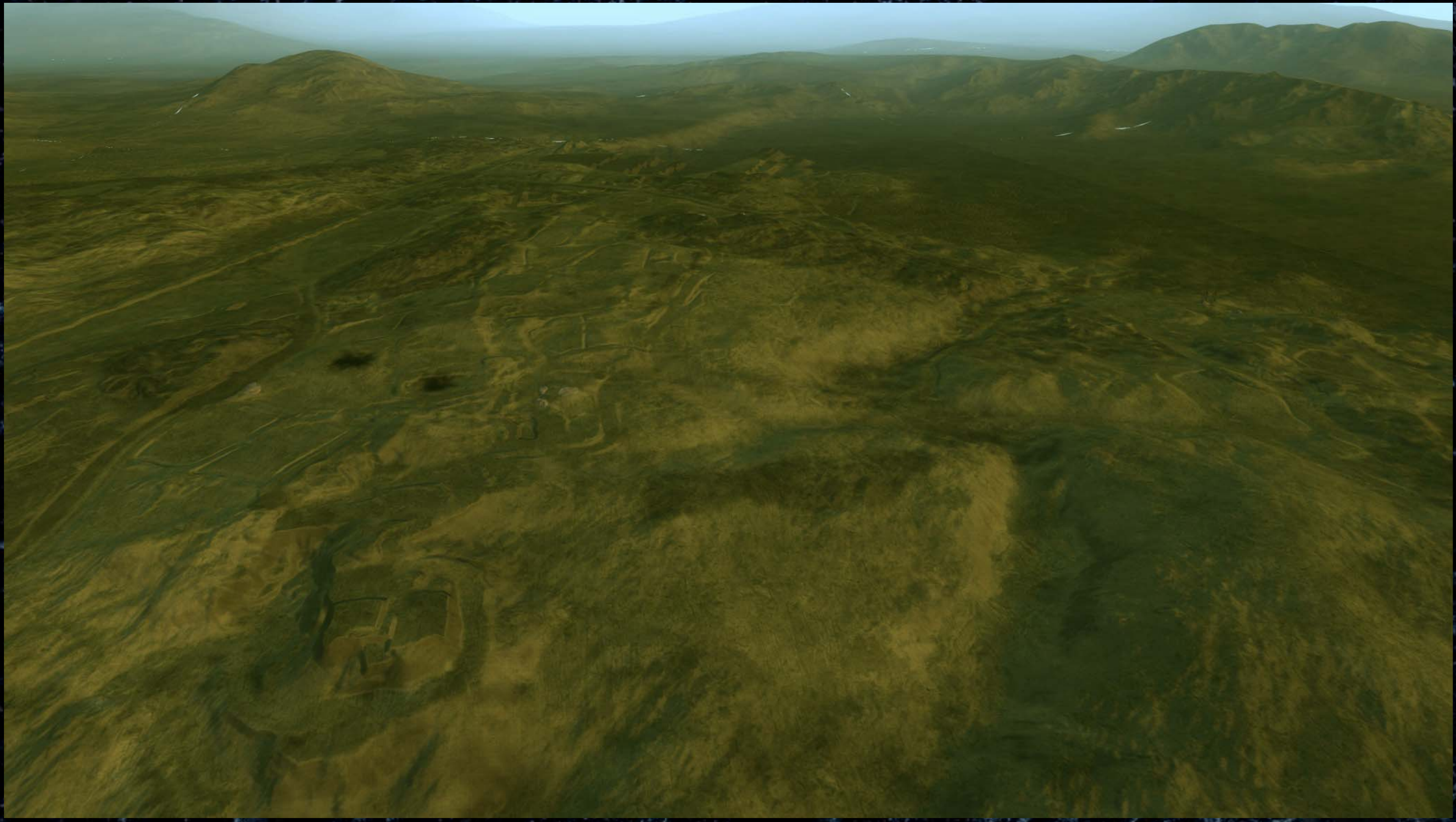
Shader compositing

- Multiple overlapping materials on terrain
- Pre-process gathers all material combinations
  - Of materials used in 16x16 m areas
- Builds big single pass shaders
  - Links together shader graphs (simple!)
  - Redundant resources & calculations automatically removed
- Dynamic flow control to avoid texture & ALU instructions for materials with mask = 0



2 materials (r & g) creating 3 combos due to overlap

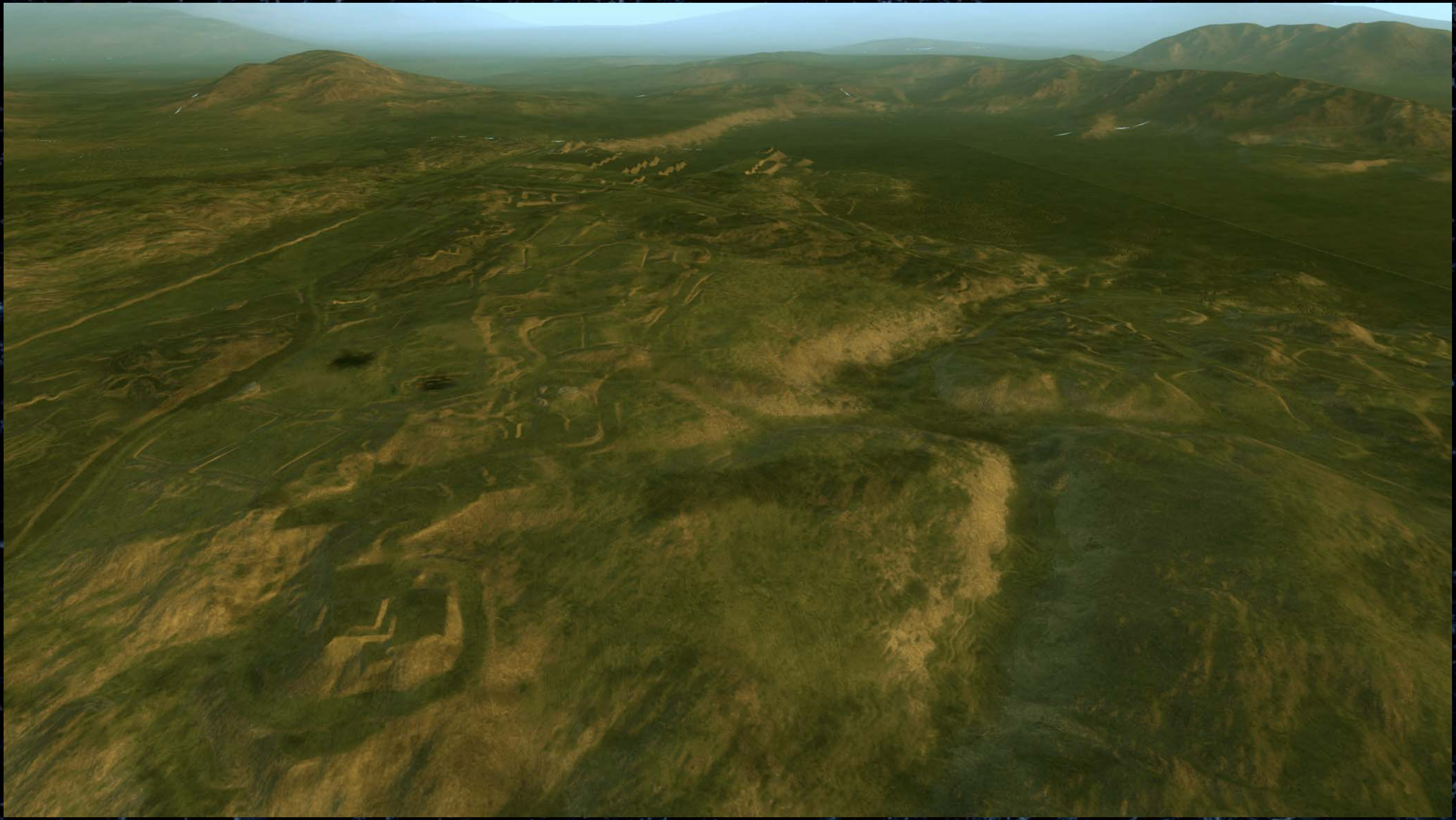
Base grass material



SIGGRAPH2007

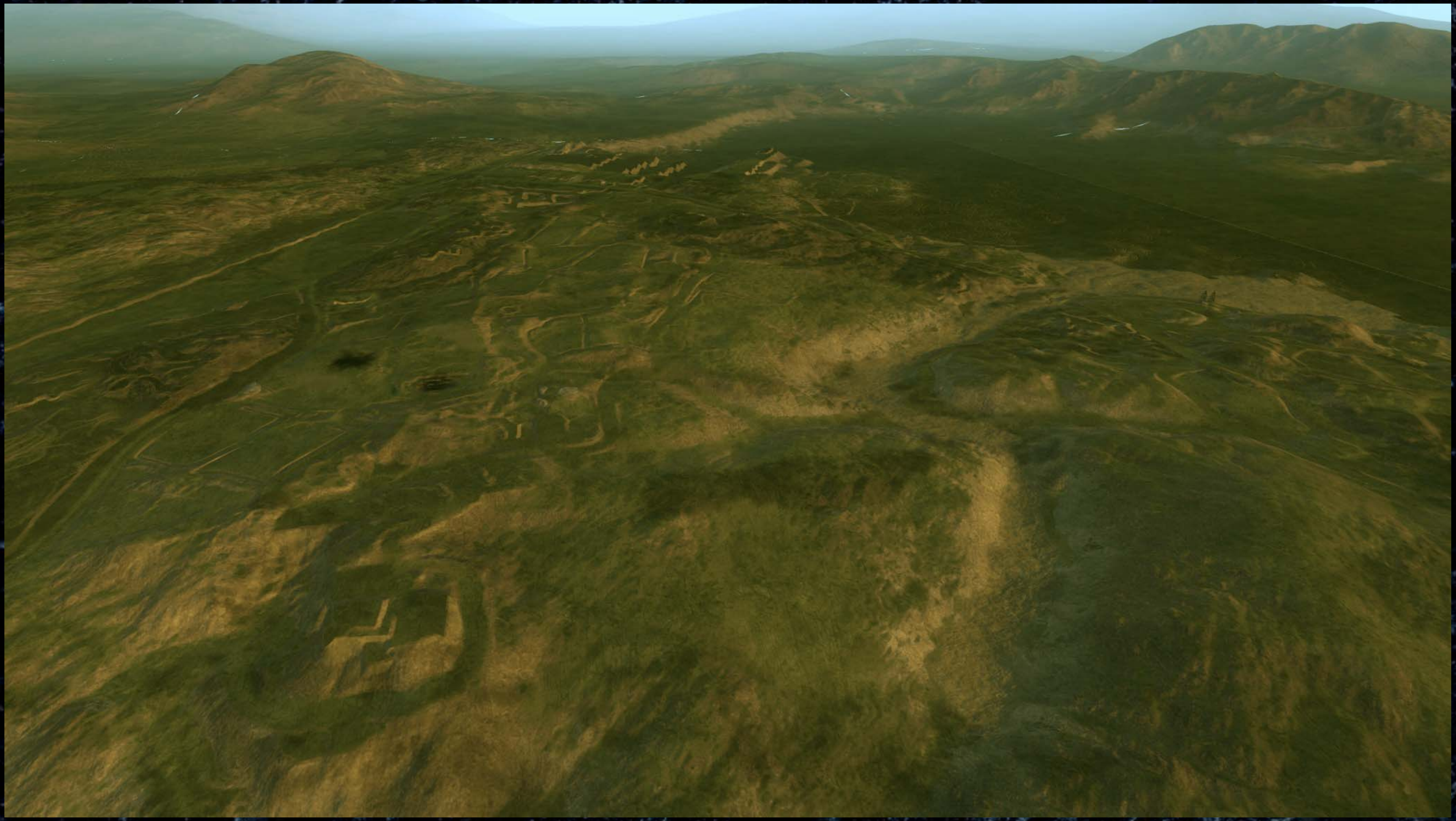
FROSTBITE
A DICE TECHNOLOGY

Dirt on slopes added

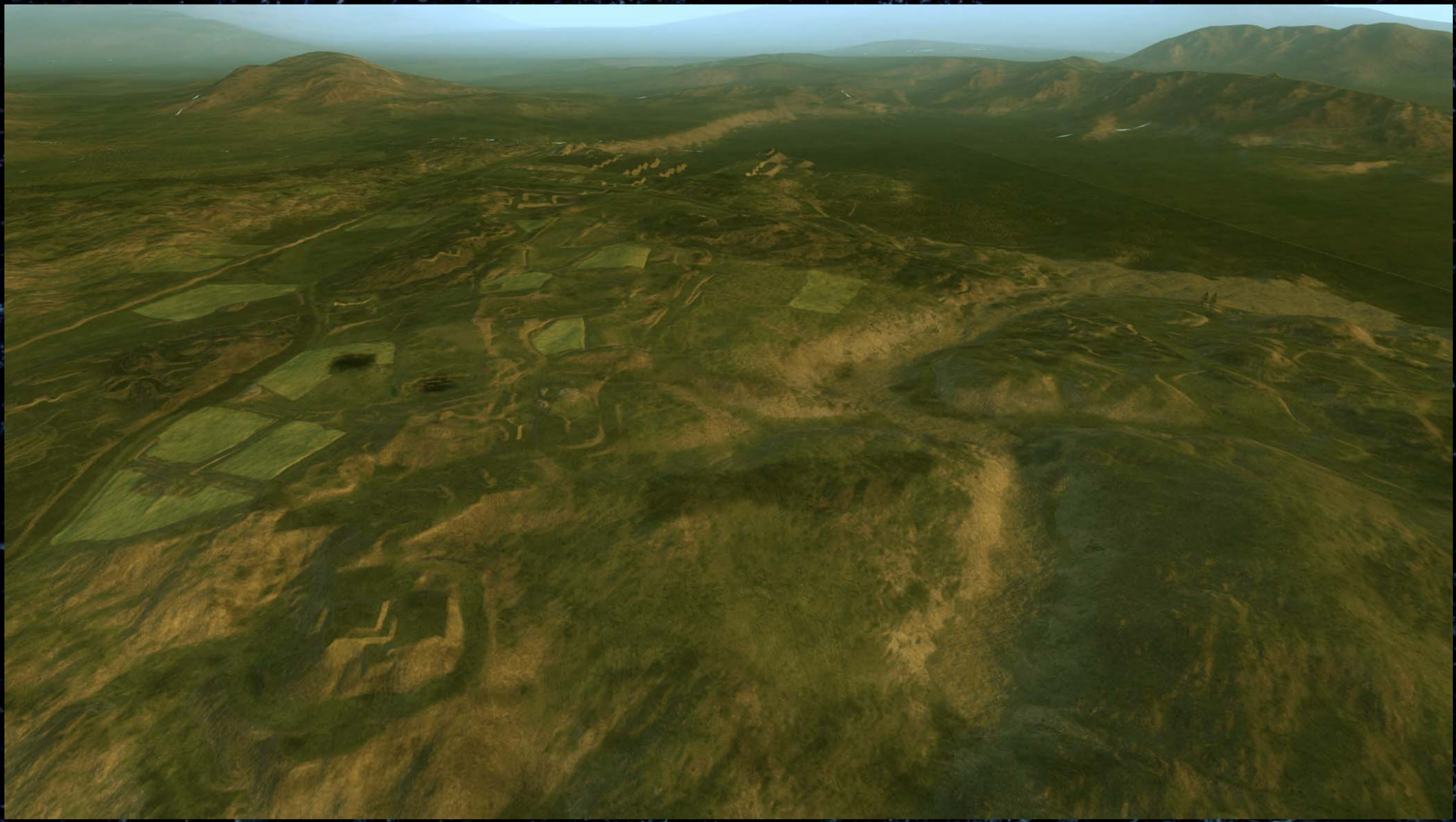


DICE SIGGRAPH2007 FROSTBITE A DICE TECHNOLOGY

Sea/river floor material added

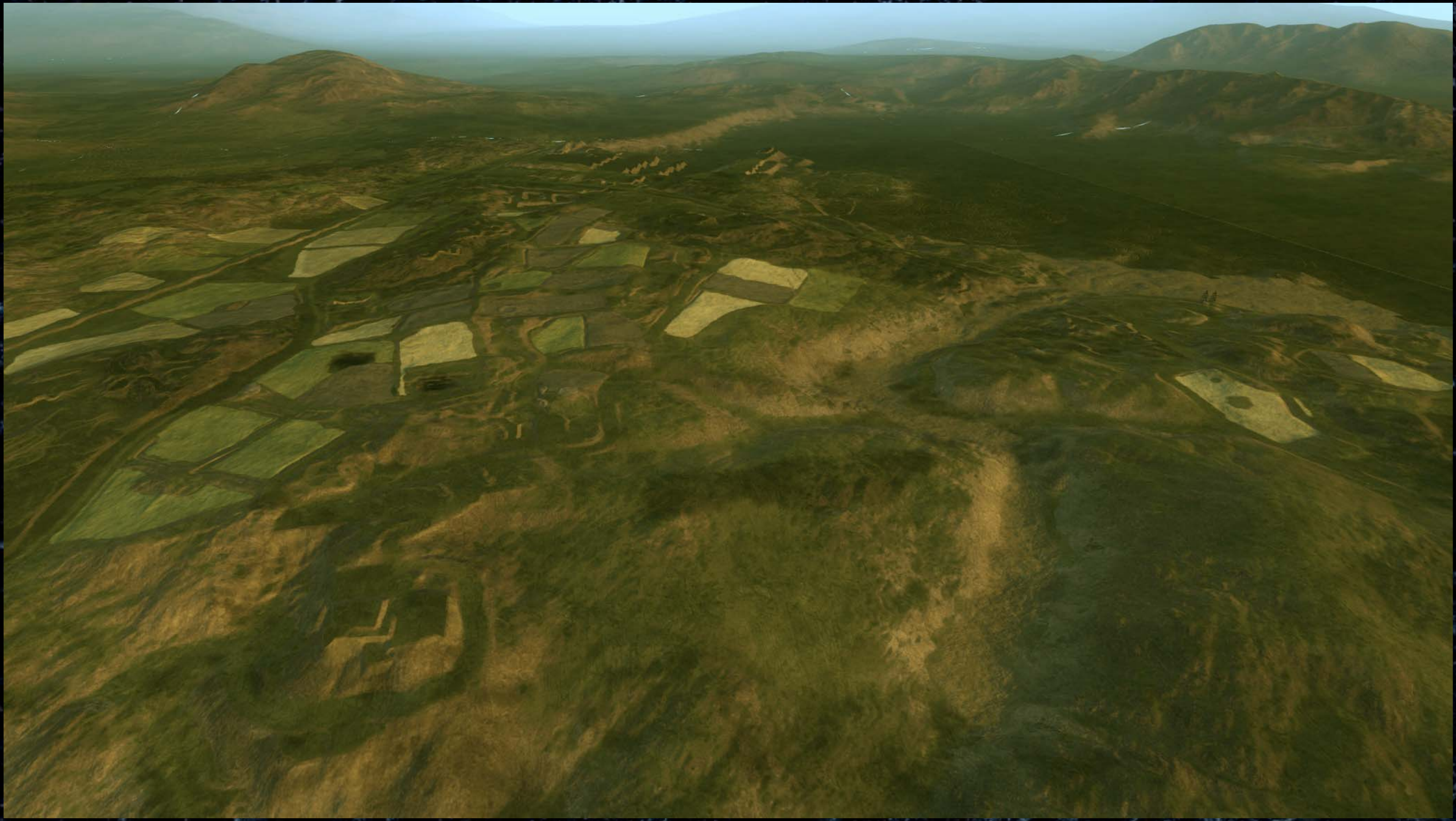


Fields added (painted masks)

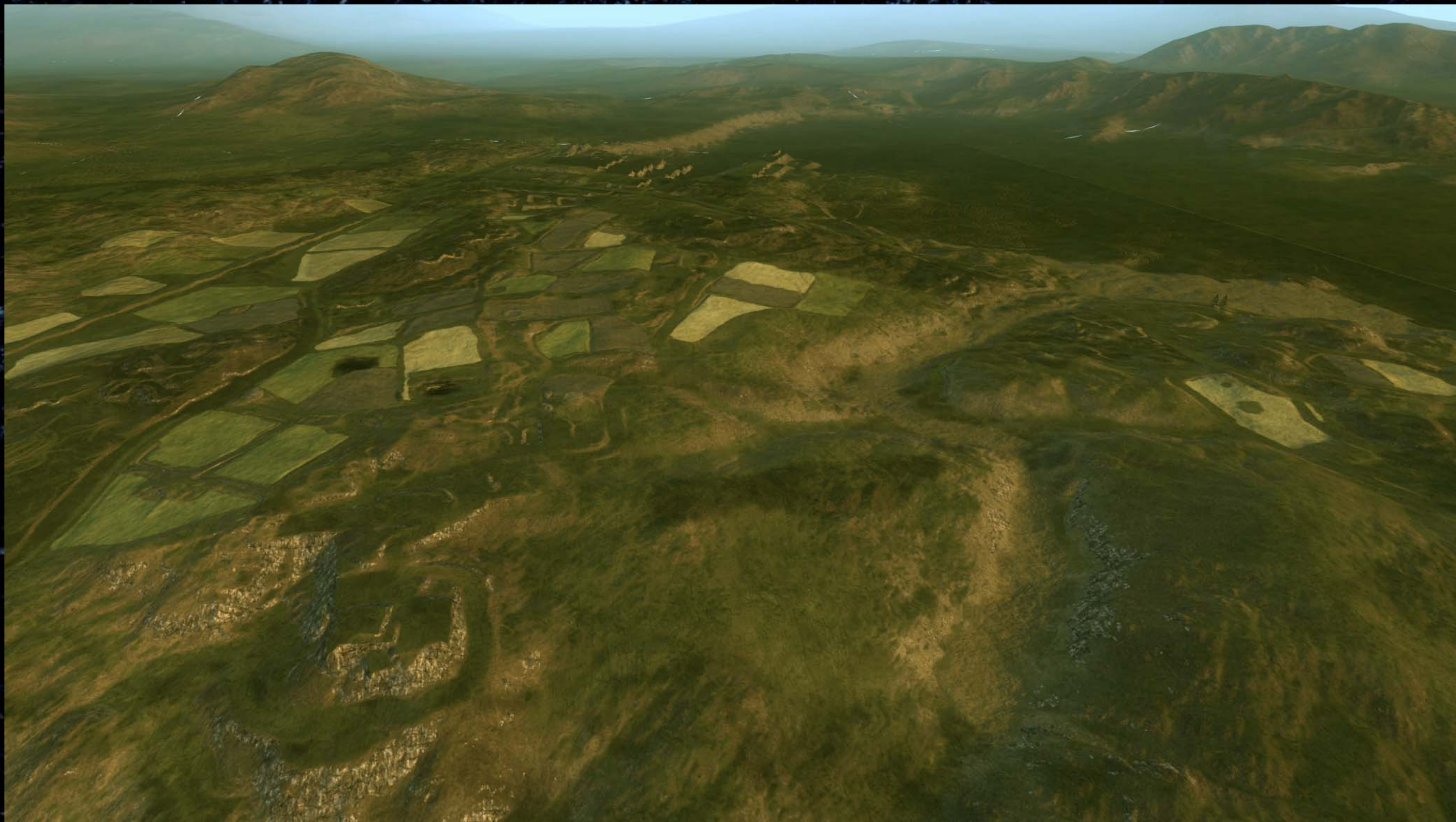


DICE SIGGRAPH2007 FROSTBITE A DICE TECHNOLOGY

2 more field types added



Slope-based cliffs added



End result. Road decals + minor materials

Outline

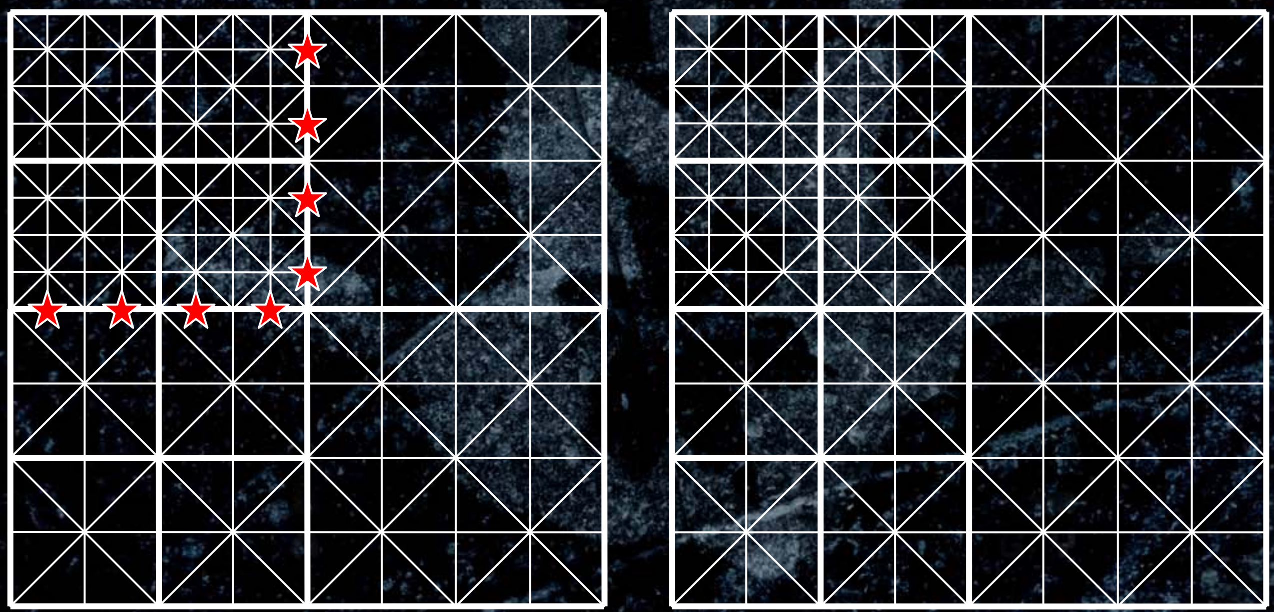
- Previous games
- Terrain overview
- Graph-based shaders
- Terrain shading & texturing
- Terrain rendering
- Undergrowth
- Conclusions
- Future

Terrain rendering

- Quadtree for culling & LOD
  - Subdivided dependent on distance
- Leaves are 33x33 fixed vertex grids
  - Simple
  - Vertex texture fetch when supported/efficient
  - CPU/SPU-filled semi-static height vertex buffer pool otherwise
- Fixed grid resolution important for ground destruction

Geometry LOD

- T-junctions between patches of different LOD
- Need to be removed, causes rendering artifacts
  - Due to vertex shader heightfield sampling

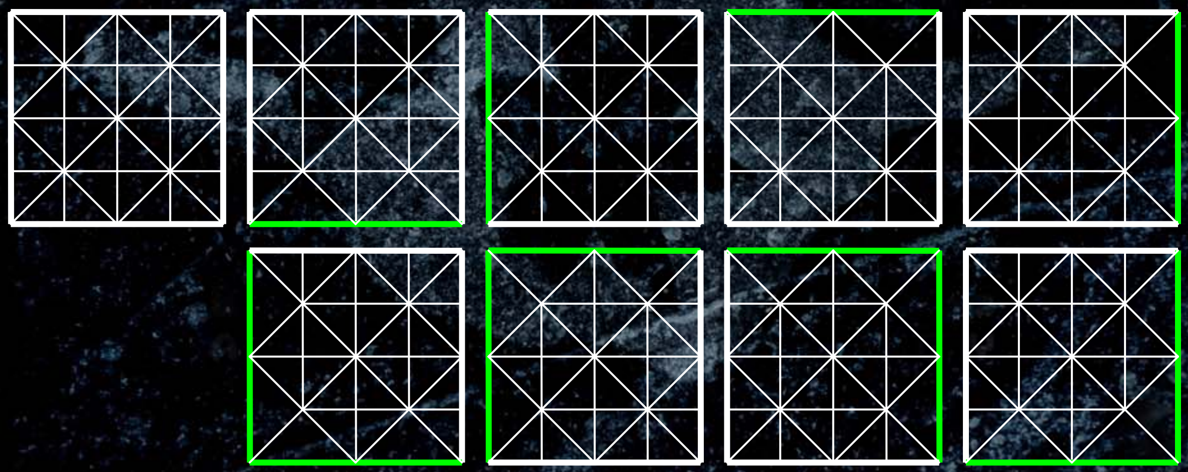


Before Removed

DICE SIGGRAPH2007 FROSTBITE A DICE TECHNOLOGY

T-junction solution

- Limit neighboring patches to max 1 level difference
- Select index buffer depending on which side borders to lower-resolution LOD
- Only 9 permutations needed



Outline

- Previous games
- Terrain overview
- Graph-based shaders
- Terrain shading & texturing
- Terrain rendering
- Undergrowth
- Conclusions
- Future

Undergrowth

- Heightfields with good texturing & shading isn't enough up close
- Need detail geometry
  - Undergrowth, small foliage, litter, debris
- Must be able to change with destruction
- Manual placement not feasible nor preferred

Undergrowth example

No undergrowth

With undergrowth

DICE SIGGRAPH2007 FROSTBITE A DICE TECHNOLOGY

Undergrowth overview (1/2)

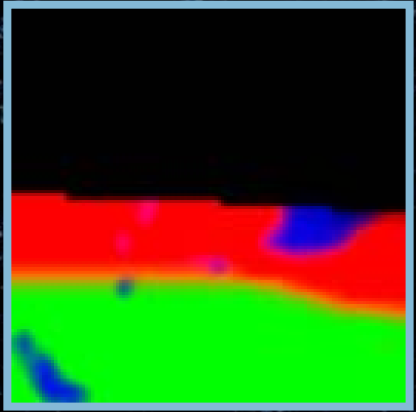
- Instance low-poly meshes around views
- Alpha-tested / alpha-to-coverage
  - Fillrate and sort-independence
- Procedural on-demand distribution
  - Using terrain materials & shaders
  - Gigabyte of memory if stored
  - Regenerate areas on destruction
  - GPU-assisted

Undergrowth overview (2/2)

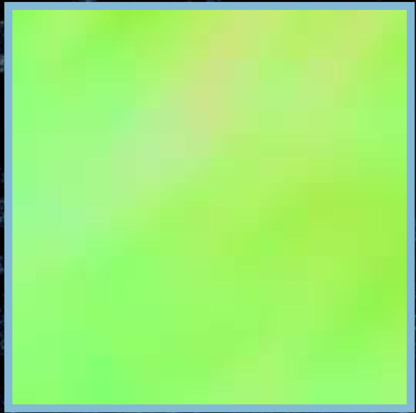
- Managed through a virtual grid structure
  - 16x16m cells
- Cells allocated from a fixed pool
  - As view position changes
- Cells contain
  - Semi-static vertex buffer with 4x3 fp16 instancing transforms
  - List of which instance uses which mesh

Undergrowth generation

- GPU renders out 4-8 terrain material masks & terrain normal
  - From the area the cell covers
  - 3x 64x64 ARGB8888 MRT
- CPU/SPU scans through texture and distributes instances
  - Fills instancing transform buffer
  - Good fit for D3D10 Stream Output



Generated mask



Normalmap

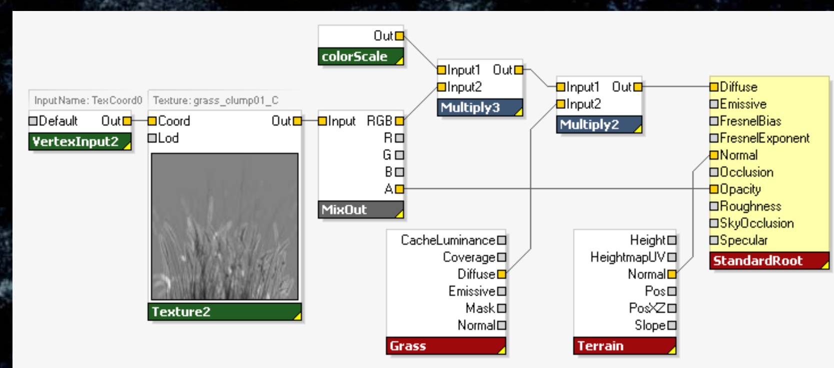
Undergrowth distribution

- Based on randomly jittered grid pattern
  - Grid size determined by material density
  - Random offsets to grid points of max half cell size
  - Gives uniform but varied distribution
  - No / controlled overlap of instances
  - Looks and performs better than fully random solution
- Deterministic results
  - Grid cell position as seed
  - Important both locally (revisit) and over network

Undergrowth rendering

- Simple instanced mesh rendering
- Uses arbitrary surface shaders
  - Unified per-pixel lighting & shadowing
  - Can use cached terrain normalmap to fit in
- Overdraw main performance bottleneck
  - Front-to-back cell sorting

Shadows on undergrowth



Undergrowth surface shader

Outline

- Previous games
- Terrain overview
- Graph-based shaders
- Terrain shading & texturing
- Terrain rendering
- Undergrowth
- Conclusions
- Future

Conclusions

- Very scalable & extendable
  - Flexible framework for performance tradeoffs
  - Low memory usage
- Higher quality but higher cost in general
  - For performance
  - For artists
    - Complex shaders requires technical artists
- Simple workflow for undergrowth
  - Huge data amplification
  - High bang for the buck

Future / Ideas

- *Very* complex surface shaders
  - ALU-based noise
  - Wang-tiles
  - More care for shader antialiasing
- Vector texture maps as masks
- Cached procedural texture generation
  - Texture synthesis
- Displacement mapping
- Adv. natural undergrowth distribution patterns
  - Fully on GPU or SPU

Questions?

Contact: johan.andersson@dice.se