Terrain in Battlefield 3:
A modern, complete and scalable system

Mattias Widmark
Software Engineer, EA Digital Illusions (DICE), Stockholm

GAME DEVELOPERS CONFERENCE
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9
2012

Overview

- Scalability – hierarchies, payloads and limitations
- Workflows – realtime in-game editing
- CPU and GPU performance
- Procedural virtual texturing – powerful GPU optimization
- Data streaming – minimizing memory footprint
- Robustness – global prioritization
- Procedural mesh generation
- Conclusions

But first…

- … what are we talking about?

- Frostbite terrain has many aspects other than the terrain mesh itself
  - Let’s look at them!

- Heightfield-based
- Mesh procedurally generated at runtime

- Surface rendering with *procedual shader splatting*
  - Arbitrary shaders splatted according to painted masks

- Spline and quad decals

- *Terrain decoration*
  - Automatic distribution of meshes (trees, rocks, grass) according to mask
  - Billboards supported

- Terrain decoration
  - Important as the terrain surface itself

- Destruction/dynamic terrain
  - Destruction depth map
    - Controls crater depth around ie static models
  - Physics material map
    - Controls surface effects, audio, crater depth and width

- Rivers/lakes
  - Implemented as free-floating decals
  - Water depth in pixel shader

Terrain raster resources

- Multiple raster resources used
  - Heightfield
  - Shader splatting masks
  - Colormap, used as an overlay on top of shader splatting
  - Physics materials
  - Destruction depth mask
  - Albedo map for bounce light
  - Additional mask channels

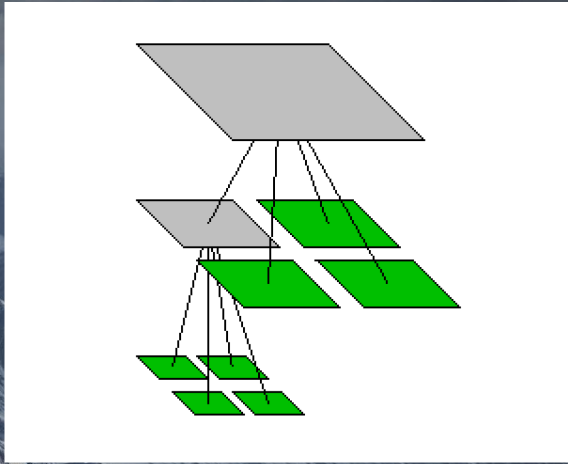
Overview

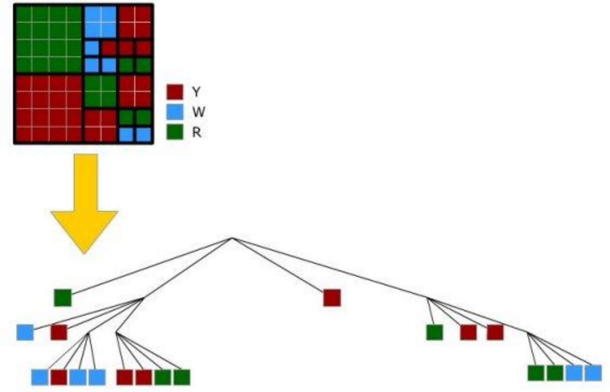
- **Scalability – hierarchies, payloads and limitations**
- Workflows – realtime in-game editing
- CPU and GPU performance
- Procedural virtual texturing – powerful GPU optimization
- Data streaming – minimizing memory footprint
- Robustness – global prioritization
- Procedural mesh generation
- Conclusions

Scalability

- Our definition of *scalability*
  - Arbitrary view distance (0.06m to 30 000m)
  - Arbitrary level of detail (0.0001m and lower)
  - Arbitrary velocity (supercars and jets)
- Main observation
  - It is all about hierarchies!
  - Consistent use of hierarchies gives scalability ”for free”
- Hierarchies not new to terrain rendering
  - Frostbite approach similar to flight simulators
- *Quadtree* hierarchies used for all spatial representations

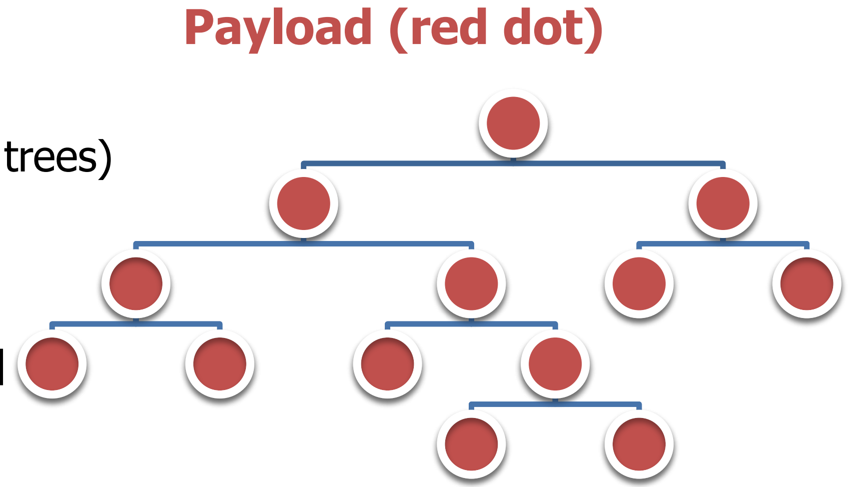
- Assuming knowledge of quadtrees, we jump right into Frostbite specifics!





Quadtree node payload

- The *node payload* is a central concept
- A quadtree node can be attributed with a ”data blob”; the payload
- Payload is
  - a tile of raster data
  - a cell of terrain decoration
    - A list of instances (rock, grass, trees)
  - a piece of decal mesh
- All nodes have payload...
  - ... but only a few have it loaded



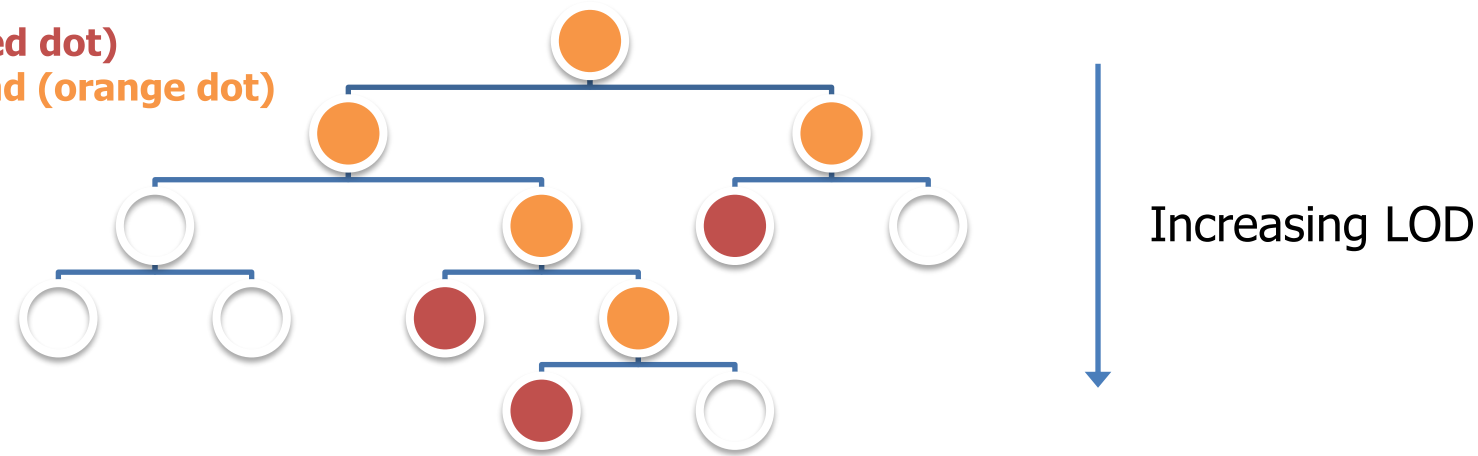
Nodes with and without payload

- Payloads are constantly in motion
  - They are loaded (streamed), generated or freed evey frame
  - Only a fraction of the nodes have payload resident
- Payload movement is governed by prioritization mechanisms...
  - ... but more of that later

LOD payload

- Non-leaf nodes have payload too
- These payloads are used as LODs
- Detail level depends on payload depth
  - Nodes closer to root represent lower detail

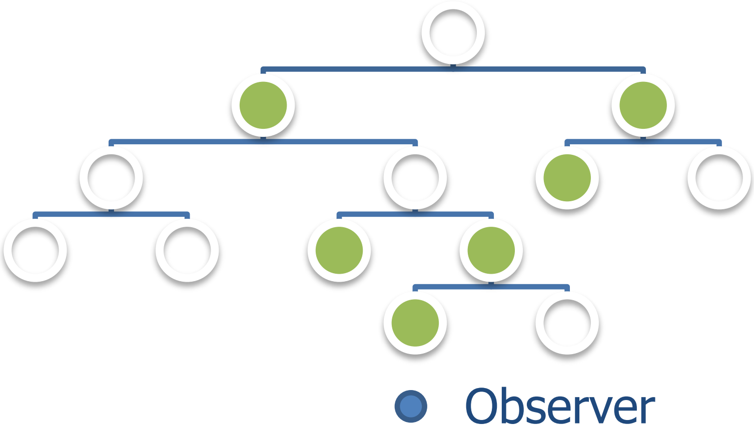
Payload (red dot)
LOD payload (orange dot)



View-dependent payload usage

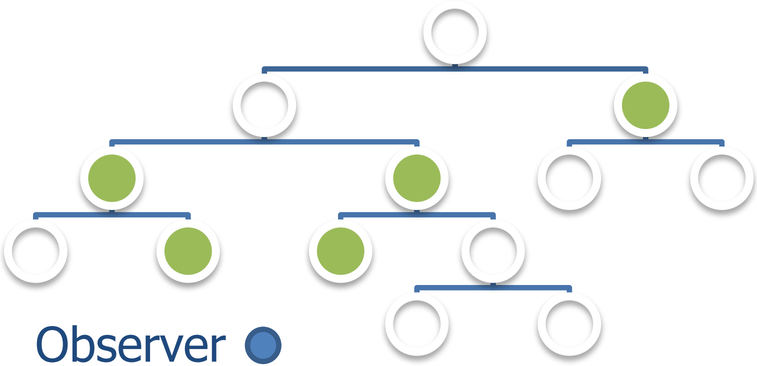
Set of payloads (green dots) used for a certain observer position

- Note area to the left is distant and use lower LOD



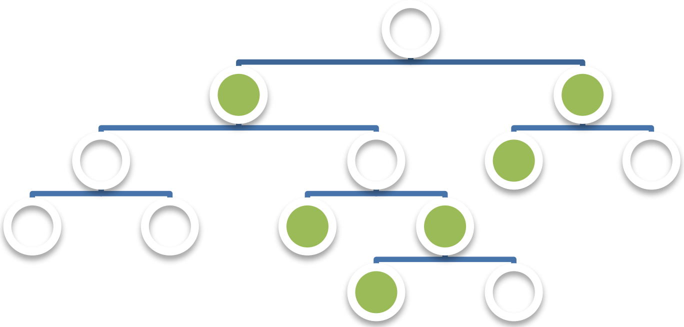
Observer moves and another set is used

- Area to the left now use higher LOD

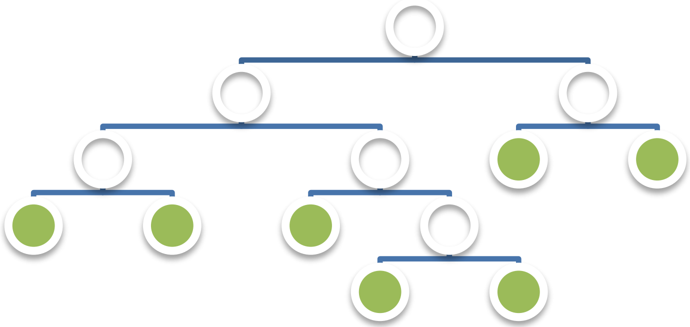


Motivation for LOD payloads

- With LOD
  - cost (payload count) is mostly independent of terrain size
  - Scalable!

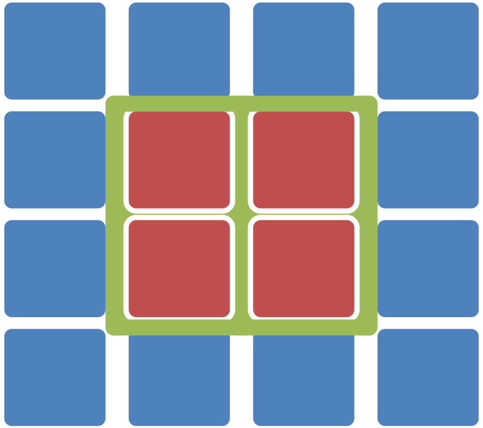


- Without LOD
  - cost depend on terrain size
  - Not scalable!



Generation of raster LOD payload

- Source data and workflows on leaf level
- LODs generated automatically by pipeline
- Requirements
  - Tile overlap (borders) for rendering algorithms
  - Continuity
- Recursive (reverse) depth first algorithm
  - Green LOD tile is generated
  - Four children (red) and up to 12 neighbor tiles (blue) are used

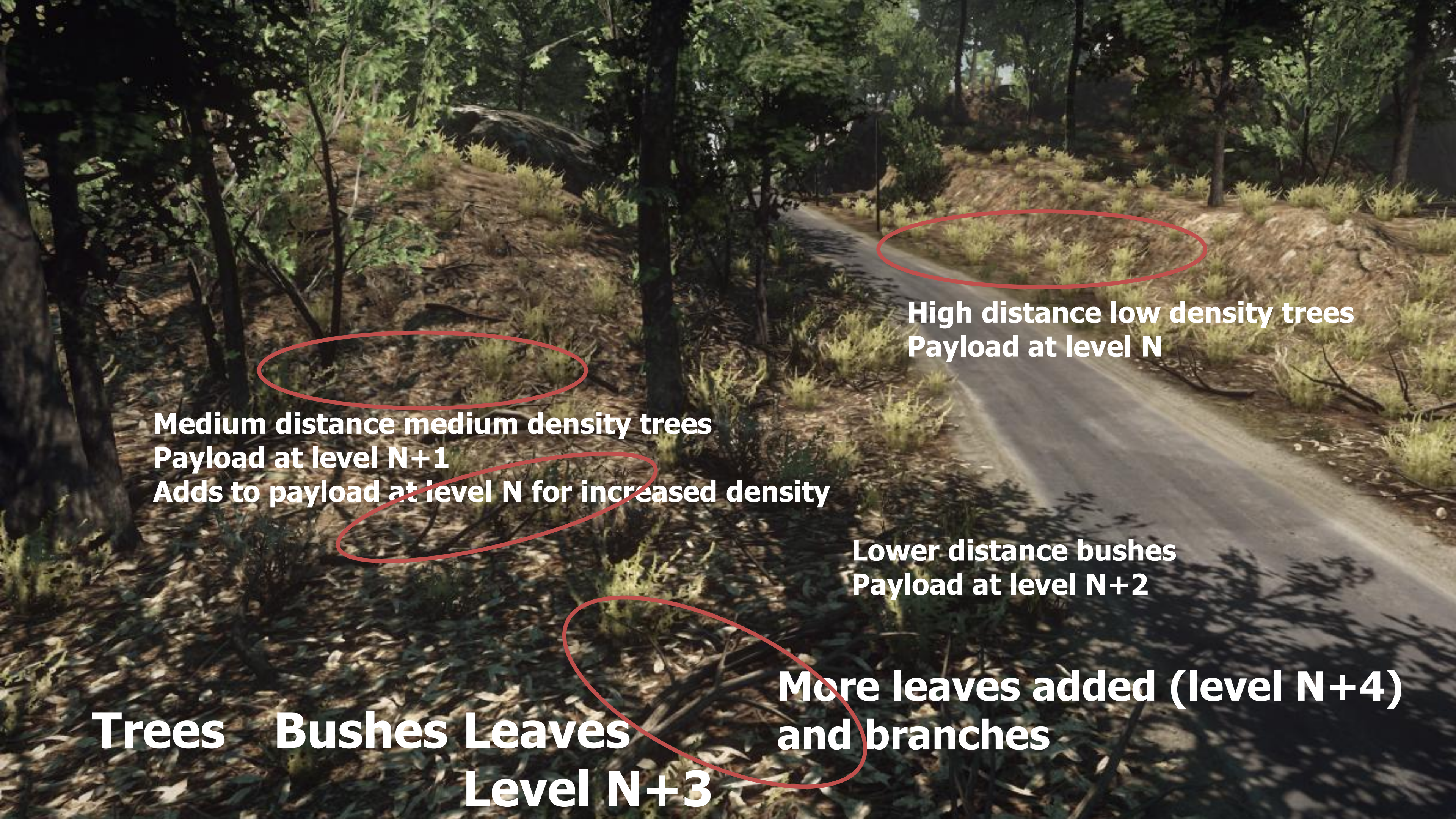


Terrain decoration payloads

- Terrain decoration payload
  - Is a list of instance transforms (for grass, trees, rocks)
  - Is generated at runtime according to
    - scattering rules
    - shader splatted masks (position/density)
      - Note that we allow shaders to modify masks!
    - heightfield (ground-clamping and orientation)

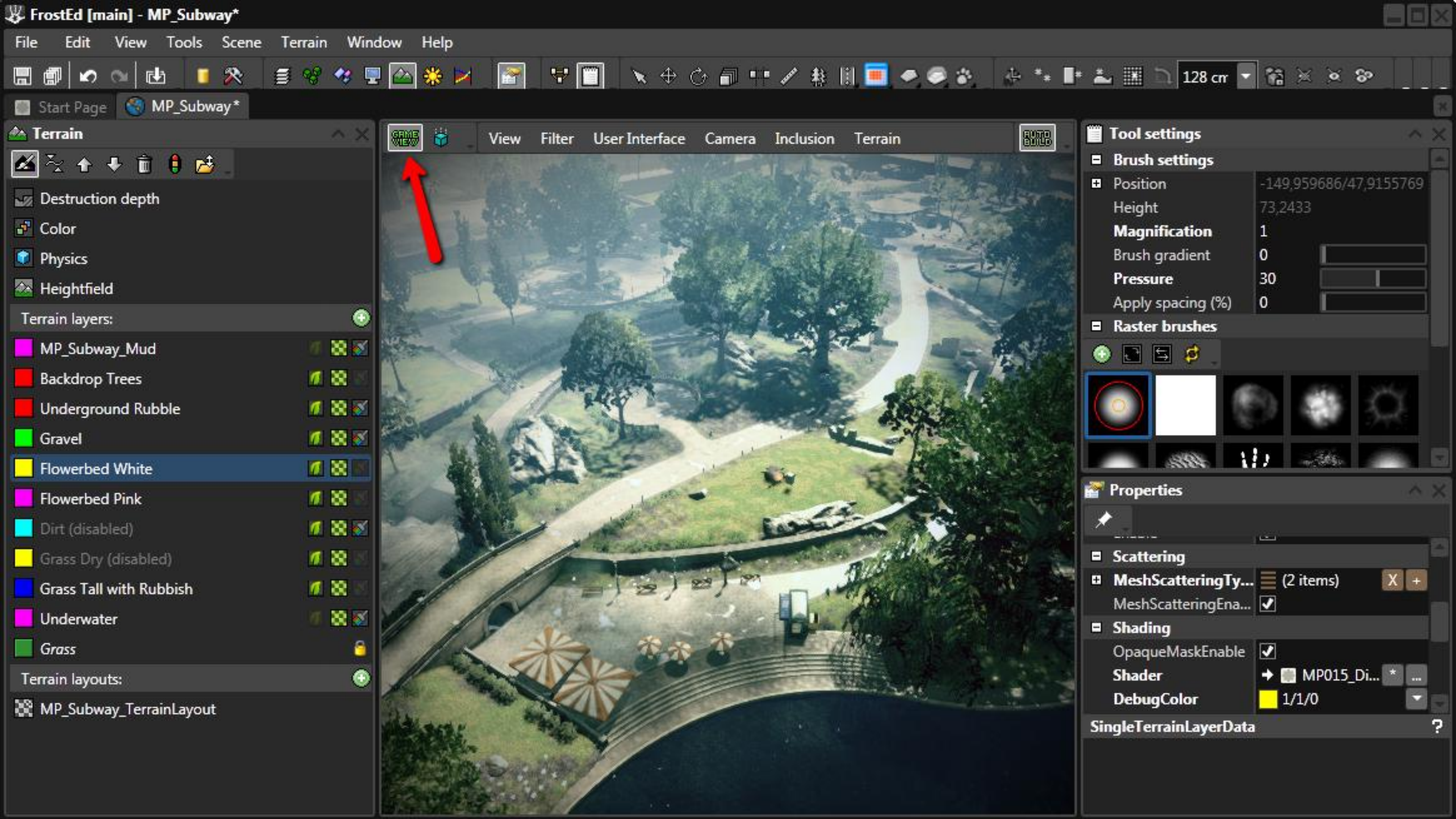
Terrain decoration LOD payloads

- Quite unique (and slightly confusing) concept
- LOD payloads used for scalability
  - Near-root payloads provide high view distance
  - Near-leaf payloads provide high density
- Payloads can overlap
  - Providing high view distance *and* density


High distance low density trees
Payload at level N
Medium distance medium density trees
Payload at level N+1
Adds to payload at level N for increased density
Lower distance bushes
Payload at level N+2
Trees
Bushes Leaves
Level N+3
More leaves added (level N+4)
and branches

Overview

- Scalability – hierarchies, payloads and limitations
- **Workflows – realtime in-game editing**
- CPU and GPU performance
- Procedural virtual texturing – powerful GPU optimization
- Data streaming – minimizing memory footprint
- Robustness – global prioritization
- Procedural mesh generation
- Conclusions



FrostEd [main] - MP_Subway*
File
Edit
View
Tools
Scene
Terrain
Window
Help
128 cm
Start Page
MP_Subway*
Terrain
Destruction depth
Color
Physics
Heightfield
Terrain layers:
MP_Subway_Mud
Backdrop Trees
Underground Rubble
Gravel
Flowerbed White
Flowerbed Pink
Dirt (disabled)
Grass Dry (disabled)
Grass Tall with Rubbish
Underwater
Grass
Terrain layouts:
MP_Subway_TerrainLayout
GAME VIEW
View
Filter
User Interface
Camera
Inclusion
Terrain
AUTO BUILD
Tool settings
Brush settings
Position
-149,959686/47,9155769
Height
73,2433
Magnification
1
Brush gradient
0
Pressure
30
Apply spacing (%)
0
Raster brushes
Properties
Scattering
MeshScatteringTy...
(2 items)
MeshScatteringEna...
Shading
OpaqueMaskEnable
Shader
MP015_Di...
DebugColor
1/1/0
SingleTerrainLayerData

External tools

- When tools are not enough terrain can be exported and imported
  - Select all or part of terrain
  - Metadata + raw file
  - Edit raw file and reimport
    - Metadata will import to right area
  - Puts *WorldMachine, GeoControl* in the loop
    - A common workflow

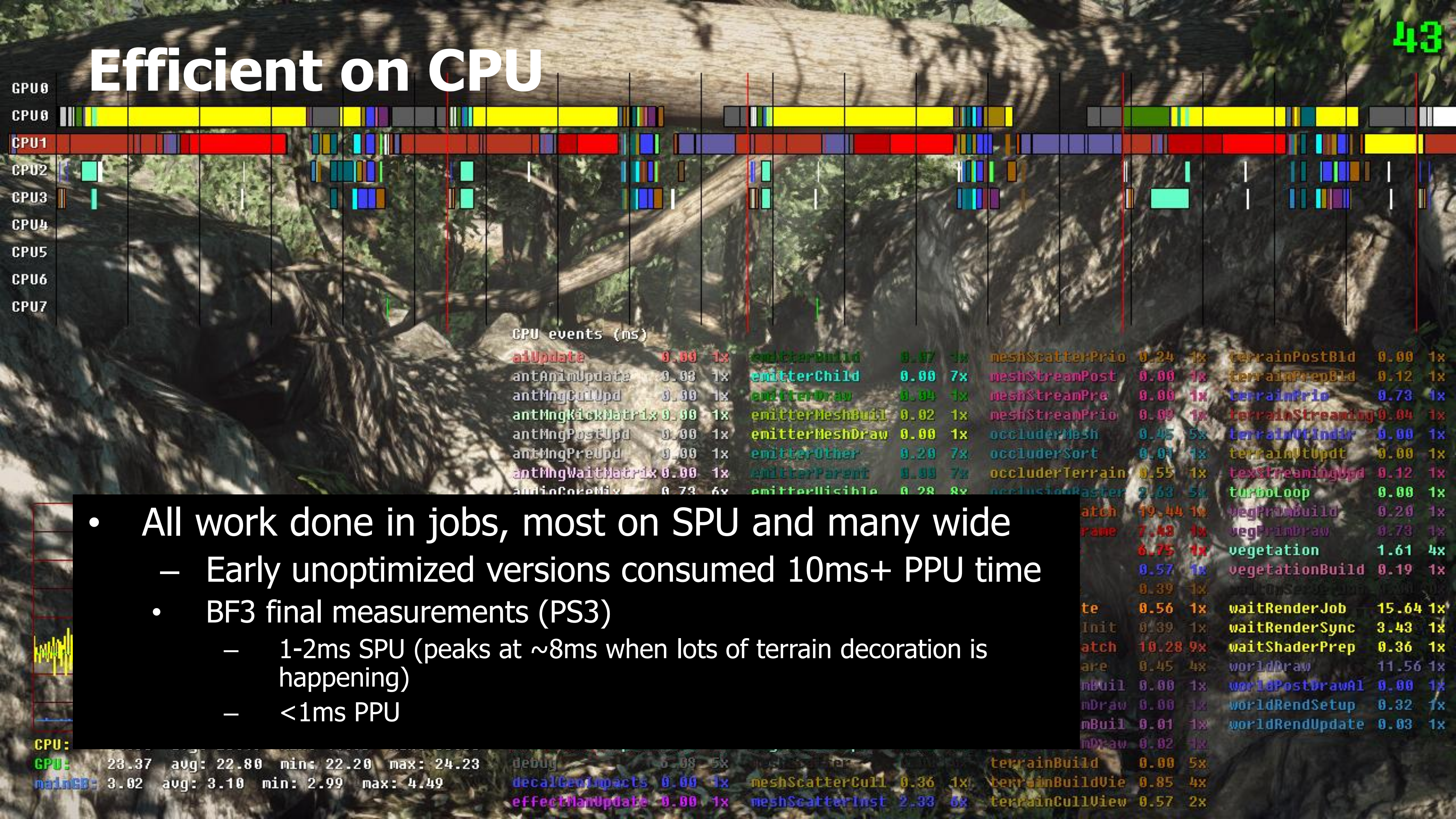
Workflow issues

- Conflict between data compression and realtime editing
- Realtime editing bypass pipeline
- Clever update scheme for procedural content needed
  - We already had one (destruction)
- Frostbite terrains too large for some popular
  - GeoControl and WorldMachine do not like 8k+

Overview

- Scalability – hierarchies, payloads and limitations
- Workflows – realtime in-game editing
- **CPU and GPU performance**
- Procedural virtual texturing – powerful GPU optimization
- Data streaming – minimizing memory footprint
- Robustness – global prioritization
- Procedural mesh generation
- Conclusions

Efficient on CPU



- All work done in jobs, most on SPU and many wide
  - Early unoptimized versions consumed 10ms+ PPU time
- BF3 final measurements (PS3)
  - 1-2ms SPU (peaks at ~8ms when lots of terrain decoration is happening)
  - <1ms PPU

Efficient on GPU

- Pre-baked shader permutations to avoid multi-pass
- *Procedual virtual texturing* exploit frame-to-frame coherency
- Typical figures (PS3):
  - Full screen GBuffer laydown (w/o *detail overlay* and terrain decoration): 2.5-3ms
  - Full screen GBuffer laydown (w/ detail overlay): 2.5-7ms
  - Terrain decoration: 1-4ms
  - Virtual texture tile compositing: 0.2-0.5ms

Overview

- Scalability – hierarchies, payloads and limitations
- Workflows – realtime in-game editing
- CPU and GPU performance
- **Procedural virtual texturing – powerful GPU optimization**
- Data streaming – minimizing memory footprint
- Robustness – global prioritization
- Procedural mesh generation
- Conclusions

GPU optimization: Procedural virtual texturing

- Motivations
  - With shader splatting, artists can create beautiful terrains…
    - … rendering very slowly (10-20ms)
  - Shader splatting not scalable in view distance
    - Cannot afford multi-pass
- By splatting into a texture
  - we leverage frame-to-frame coherency (performance)
  - can render in multiple passes (scalability)
- Rendering full screen using the texture cost 2.5-3ms (PS3)

Virtual texture key values

- 32 samples per meter
- 256x256 tiles with integrated two pixel border
- Atlas storage with default size 4k x 2k
- Two DXT5 textures

R	G	B	A	R	G	B	A
Diffuse R	Diffuse G	Diffuse B	Normal X	Smoothness/ Destruction	Normal Y	Specular	Normal Z

- Very large, can easily reach 1M x 1M (= 1Tpixel)!
  - Typical virtual textures are 64k x 64k

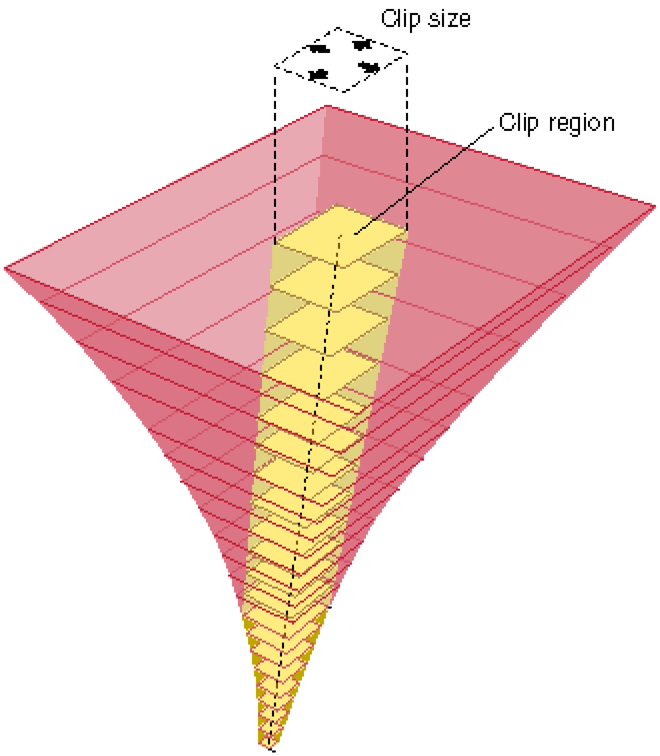
Indirection texture format

R	G	B	A
Index X	Index Y	Scale	CLOD fade

- RGBA8
- Indices into virtual texture tile atlas
- Scale factor for low-res areas...
  - ... where a tile covers more than one indirection sample
- CLOD fade factor
  - Used to smoothly fade in a newly composited tile (*fade-to* tile)
  - Previous LOD (*fade-from* tile) is already in atlas and fetched using indirection mips
  - CLOD factor updated each frame

The "Teratexture"

- How do we reach 1M x 1M?
  - Indirection texture can easily go 4k x 4k
    - Way too large!
- *Clipmap* indirection texture
  - Clipmap – early virtual texture implementation
  - Replace 4k indirection texture with 6 64x64 clipmap layers



Clipmap indirection

- Clipmap level is resolved on CPU for each draw call
  - Avoids additional pixel shader logic
  - Requires each 64x64 map to have its own mip chain
- Texture space has to be roughly organized with world space
  - Not an issue for terrain
  - More generic use cases are probably better off with multiple classic virtual textures

Tile compositing

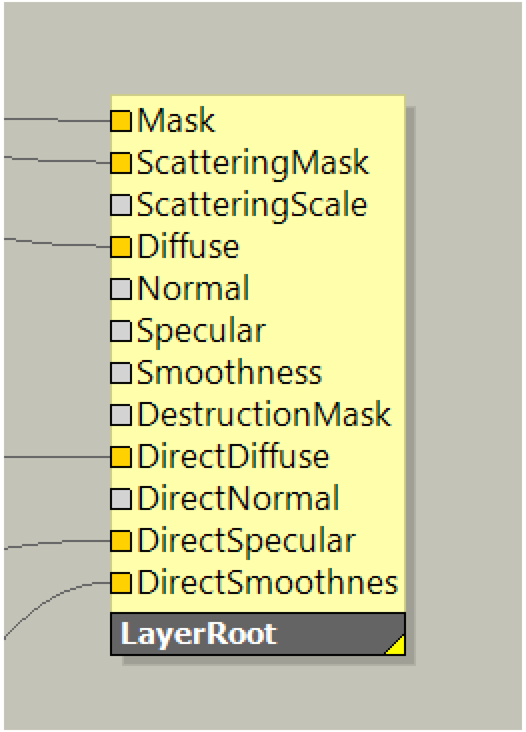
- Tiles are composited on GPU and compressed on GPU or SPU
- Benefits (compared to streaming from disc)
  - Small disc footprint – data magnification
    - Source raster data magnified ~1000x
  - Low latency
    - Tile is ready to use next frame
  - Dynamic updates
    - Destruction
    - Realtime editing
  - Efficient workflow
    - Artists don’t have to paint hundreds of square kilometers down to every last pebble

Virtual texture issues

- Blurriness (consoles have too small tile pool)
- Runtime texture compression quality
- GPU tile compositing cost offset some of the gain
  - ~0.25ms/frame on Xenon
- Limited hardware filtering support
  - Expensive software and/or lower-quality filtering
- Compositing latency
- Tile compositing render target memory usage

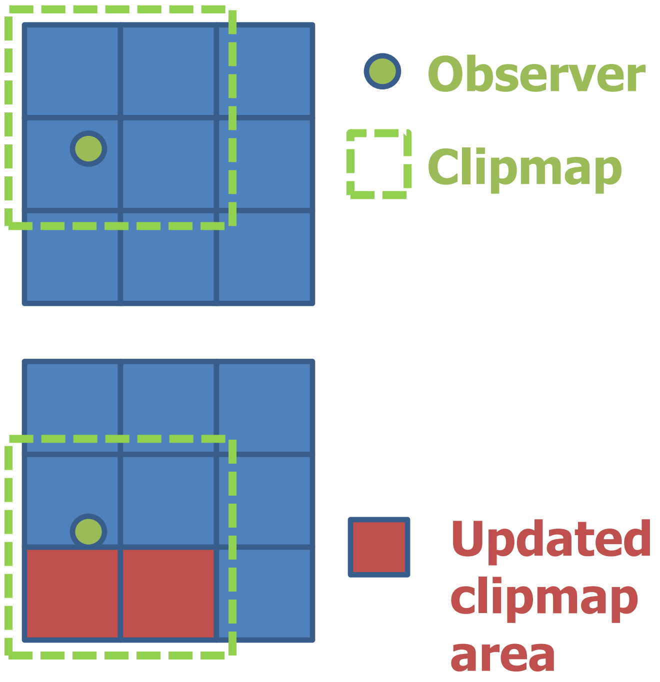
Virtual texture issues

- Virtual texture has practical limit at around 32 samples per meter
  - *Detail shader splatting* fills in to required sharpness (500-1000 samples per meter)
- We now have two shader splatting methods
  - Diffuse/Normal/Specular/Smoothness splats into virtual texture
  - Direct* splats details into Gbuffer
- Performance concerns
  - Have to limit detail view distance
  - Typically 50m-100m



Virtual texture issues

- Indirection update performance
  - Updating indirection maps is expensive
    - 4-6 64x64 maps with mips
  - Do in job (SPU)
  - Update only dirty areas
    - New tile
    - CLOD fade factor
    - Recentering when clipmap region moves
  - Wraparound
    - Only edges affected on recentering (bottom image)



Overview

- Scalability – hierarchies, payloads and limitations
- Workflows – realtime in-game editing
- CPU and GPU performance
- Procedural virtual texturing – powerful GPU optimization
- **Data streaming – minimizing memory footprint**
- Robustness – global prioritization
- Procedural mesh generation
- Conclusions

Data streaming

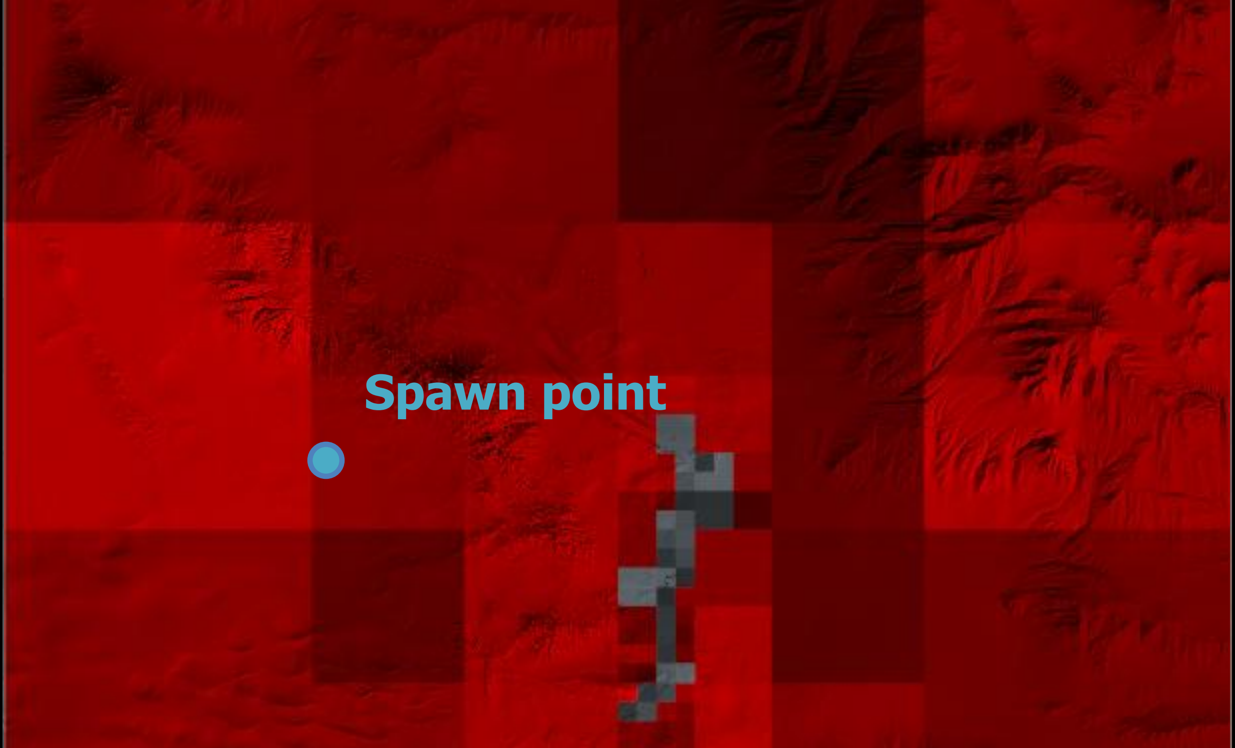
- Frostbite 1 did not stream terrain
- Streaming required for larger *Battlefield 3* and *NFS: The Run* levels

Streaming basics

- Streaming unit: Raster tiles (aka node payloads)
  - Typical tile sizes
    - Heightfield: 133x133x2 bytes
    - Mask: 66x66x1 bytes x 0-50 tiles per payload
    - Color: 264x264x0.5 bytes
- Fixed-size tile pools (atlases)
  - Typical atlas sizes
    - Heightfield: 2048x2048
    - Mask: 2048x1024
    - Color: 2048x2048

Streaming modes

- Tile-by-tile (aka *free*) streaming
  - For slower gameplay
- Tile *bundle* (aka *push-based*) streaming
  - For faster games
  - Tiles associated with a layout are bundled
  - Based on *terrain resolution layouts*
- Hybrid streaming (most common)
  - Bundles used at selected spawn points and transitions
  - Free streaming fill in the rest



Layout of all data on level

Data subset loaded at spawn point

A *Terrain resolution layout* defines the subset

Subsets loaded (and unloaded) as user progresses through level

Resident set size: Powerful *blurriness*

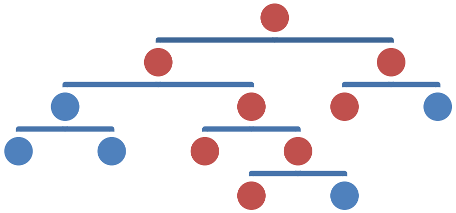
- Streaming does not remove memory footprint
  - A *resident set* is still needed
- Resident set can be huge
  - Theoretical value for BF3 level on PS3: 70+Mb!
- *Blurriness* feature shrinks resident set significantly
  - Increase blurriness by one and save around 70%
  - Very important memory saver for BF3
  - Shipped with 32Mb terrain resident set

Blurriness

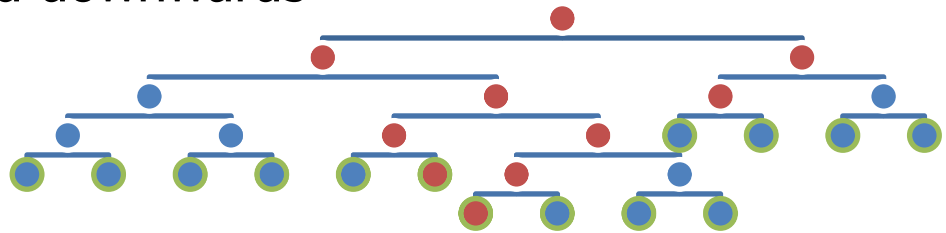
- Blurriness is mip bias applied to terrain raster streaming
  - Blurriness = 0:
    - Streaming continues until raster is sharp
  - Blurriness = 1:
    - Streaming stops when raster is slightly blurry (texel covers 2x2 pixels)
  - Blurriness = 2:
    - Streaming stops when raster is significantly blurry (texel covers 4x4 pixels)

Blurriness: Implementation by pipeline trick

- For each level of blurriness
  - cut tile size in half
  - add one leaf level
- No data is lost – it is only shifted downwards



Blurriness 0
256x256-sized tiles
8 resident tiles (500kpixel)
13 nodes



Blurriness 1
128x128-sized tiles
14 nodes added
10 resident tiles (164kpixel, saves 68%)
27 nodes

BF3 blurriness use case

- Blur expensive rasters
  - Heightfield (2 bytes per sample)
  - Mask (1 byte per sample)
- Keep cheap rasters
  - Colormap (DXT1, 0.5byte per sample)
- Put detail (for example occlusion) in colormap to hide blurry heightfield/normal map

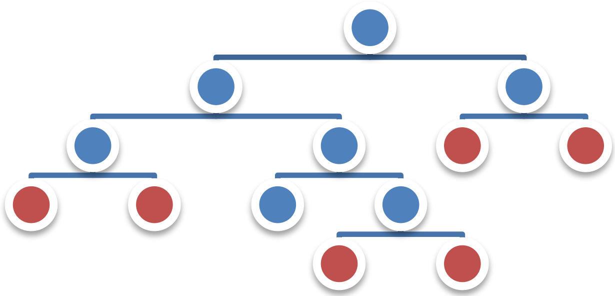
- **Physics**
  - **Wrong streaming setup gives strange artifacts**
    - **Vehicles spawning in air or in ground**
    - **Invisible and disappearing walls and holes**

Reducing disc seeks

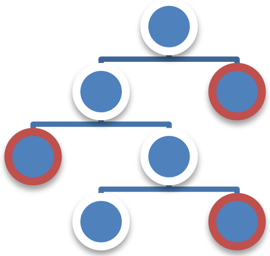
- Often main reason for latency
  - Can seek maybe four files per second
  - A terrain can have hundreds of files (tiles/payloads)
- Methods to reduce the number of seeks
  - Use terrain resolution layouts at spawn
    - Otherwise **minutes** can pass before stabilization (waiting for file seeks)!
  - Co-locate overlapping tiles of different types
    - Store heightfield tile together with color and mask tiles

Reducing disc seeks

- Methods to reduce the number of seeks (cont'd)
  - Co-locate nearby tiles
    - Group leaf node payloads as *second LOD* in ancestor node
    - Saves 20-50% seeks in typical scenario



Full dataset require 13 seeks
Leaf nodes subject for move in red



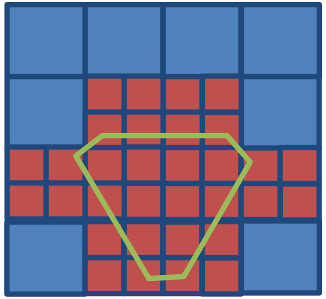
Second LOD stored in parent nodes
Full dataset is now 10 seeks

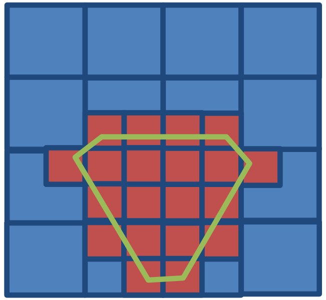
Improving throughput

- Data tiles are compressed by pipeline
  - Color tiles are optionally DXT1-compressed
  - Mask compression
    - Source tiles (256x256) are chopped up into smaller pieces (66x66)
    - Empty tile culling
    - Identical (constant value) tile merging
  - Physics materials and destruction depth are packed to four bits and RLE-compressed
  - All tiles are z-compressed

Improving throughput

- A quadtree node has zero *or* four children
- Incomplete quadtree
  - We allow zero *through* four payload children
  - Reduces bundle size by some 20%+





Latency under the rug

- Even with mentioned improvements, streaming is not instant
- General ways of hiding latency
  - CLOD to smooth most streaming LOD transitions
  - Global prioritization helps distribute punishment

Overview

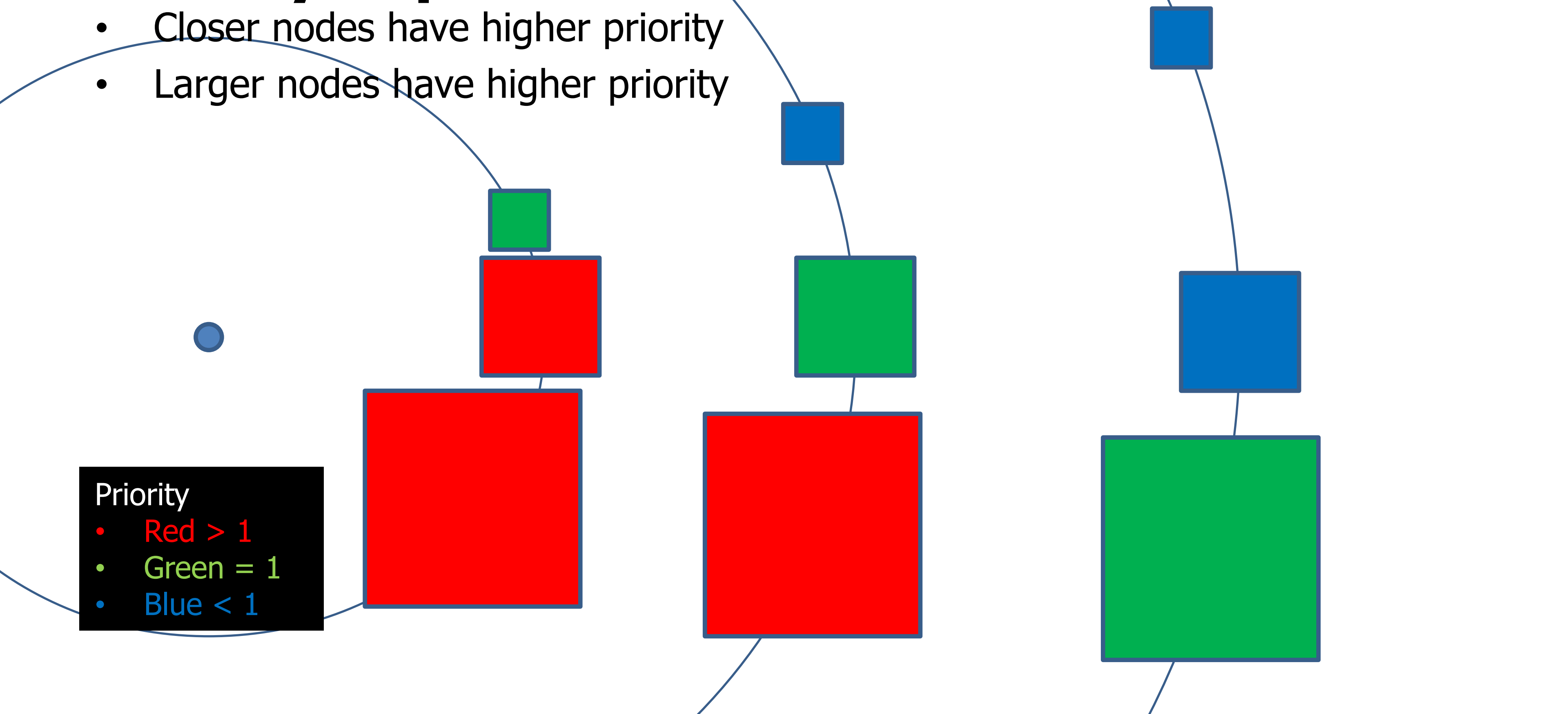
- Scalability – hierarchies, payloads and limitations
- Workflows – realtime in-game editing
- CPU and GPU performance
- Procedural virtual texturing – powerful GPU optimization
- Data streaming – minimizing memory footprint
- **Robustness – global prioritization**
- Procedural mesh generation
- Conclusions

Global prioritization

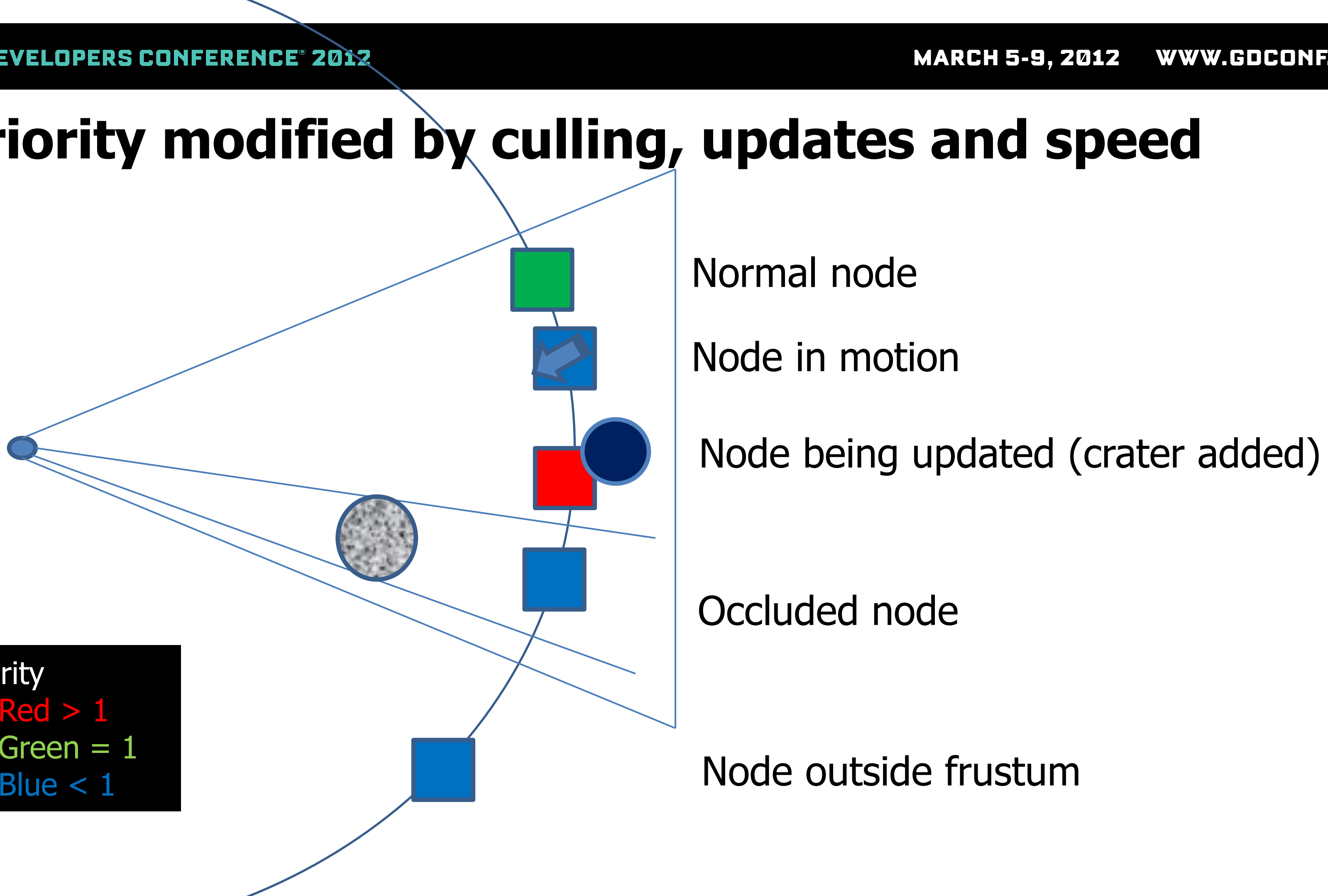
- Compute priority for each frame and each quadtree node
- Update streaming, virtual texturing and terrain decoration
  - According to priority
- Priority value
  - 1.0: On target
    - One pixel per texel
    - Terrain decoration at specified view distance
  - < 1.0: Node doesn’t need payload
  - > 1.0: Node need payload

Priority depends on distance and size

- Closer nodes have higher priority
- Larger nodes have higher priority

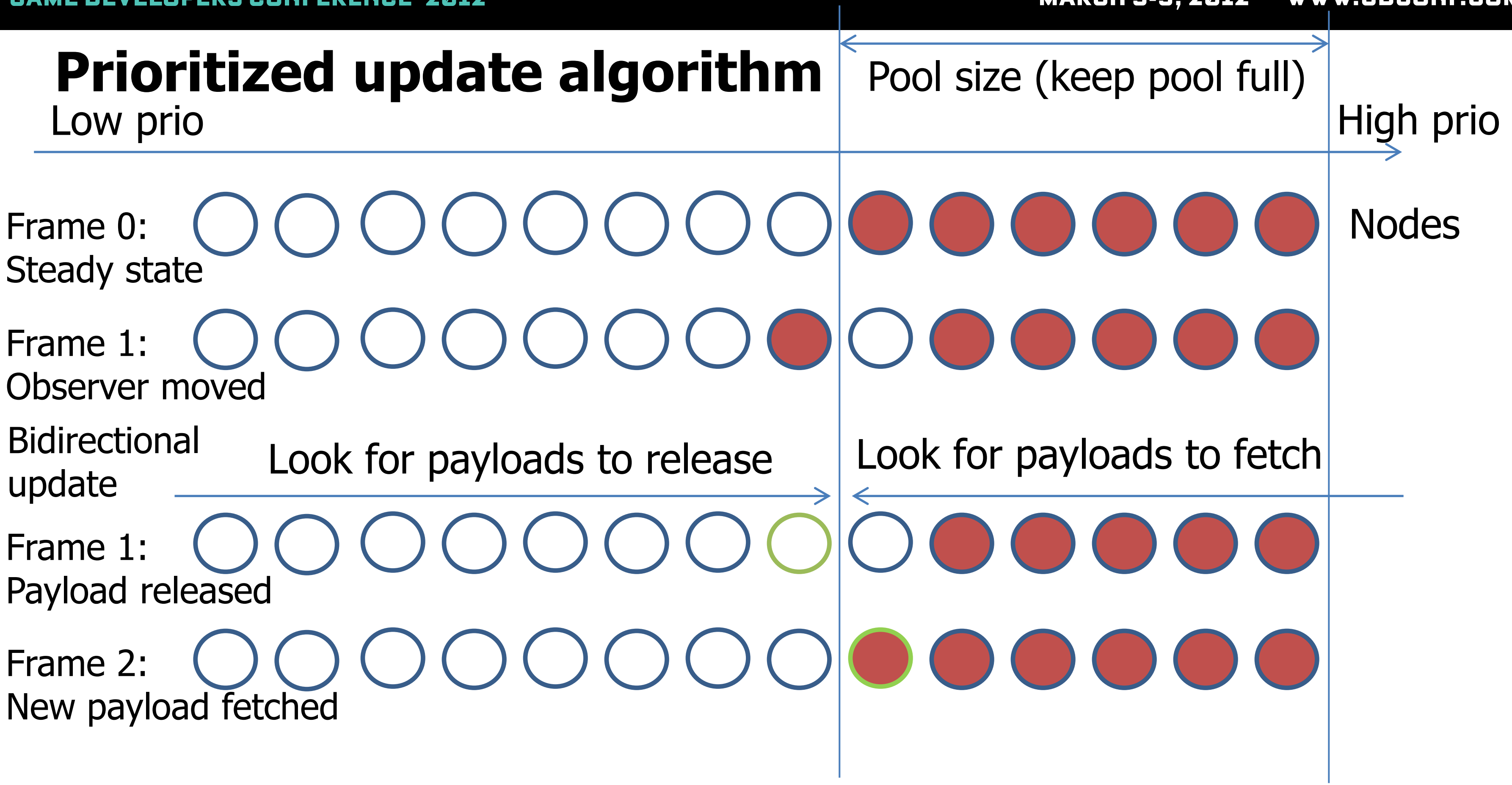


Priority modified by culling, updates and speed



Priority
- Red > 1
- Green = 1
- Blue < 1

Prioritized update algorithm



Prioritized update cost

- Priority evaluation and sorting done on SPU
  - <1ms
- Update done on PPU
  - <0.5ms

Overview

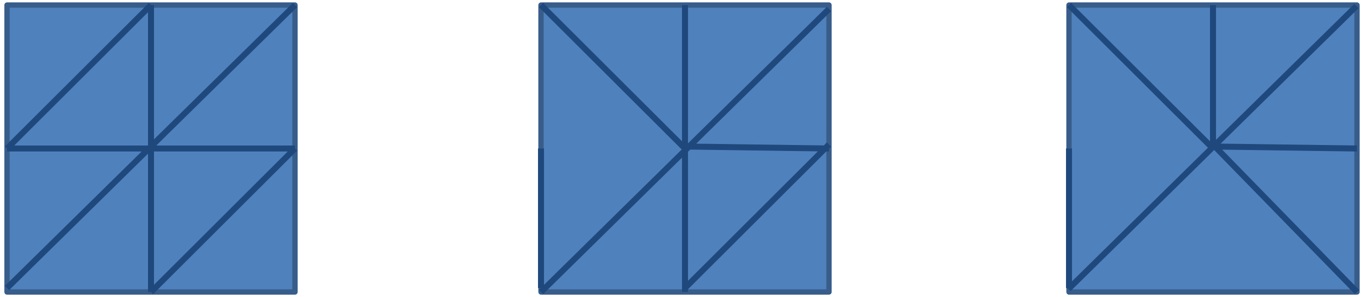
- Scalability – hierarchies, payloads and limitations
- Workflows – realtime in-game editing
- CPU and GPU performance
- Procedural virtual texturing – powerful GPU optimization
- Data streaming – minimizing memory footprint
- Robustness – global prioritization
- **Procedural mesh generation**
- Conclusions

Mesh

- Mesh generated from heightfield
- Straight-forward tessellation
  - Rendered in patches of 16x16 triangle pairs
  - Projected triangle size rougly constant - support destruction everywhere
  - Blocky crestlines (on console)
- PC, Xenon:
  - Heightfield sampled in vertex shader
- PS3:
  - Vertex shader texture fetch is too slow
    - Heightfield samples stored in vertex attribute
    - Heightfield sampled in pixel shader

Mesh stitching

- A job analyze mesh quadtree and detect LOD switch edges
- Edges are stitched by index permutations
  - Vertices are unchanged



DX11 tessellation

- Displacement mapping from heightfield
  - No additional memory needs (heightfield used as normal map)
- Straightforward hull and domain shaders
- Tessellation factor derived from projected patch edge bounding sphere
  - Tries to maintain a constant screen-space triangle width


Displacement mapping ON

Overview

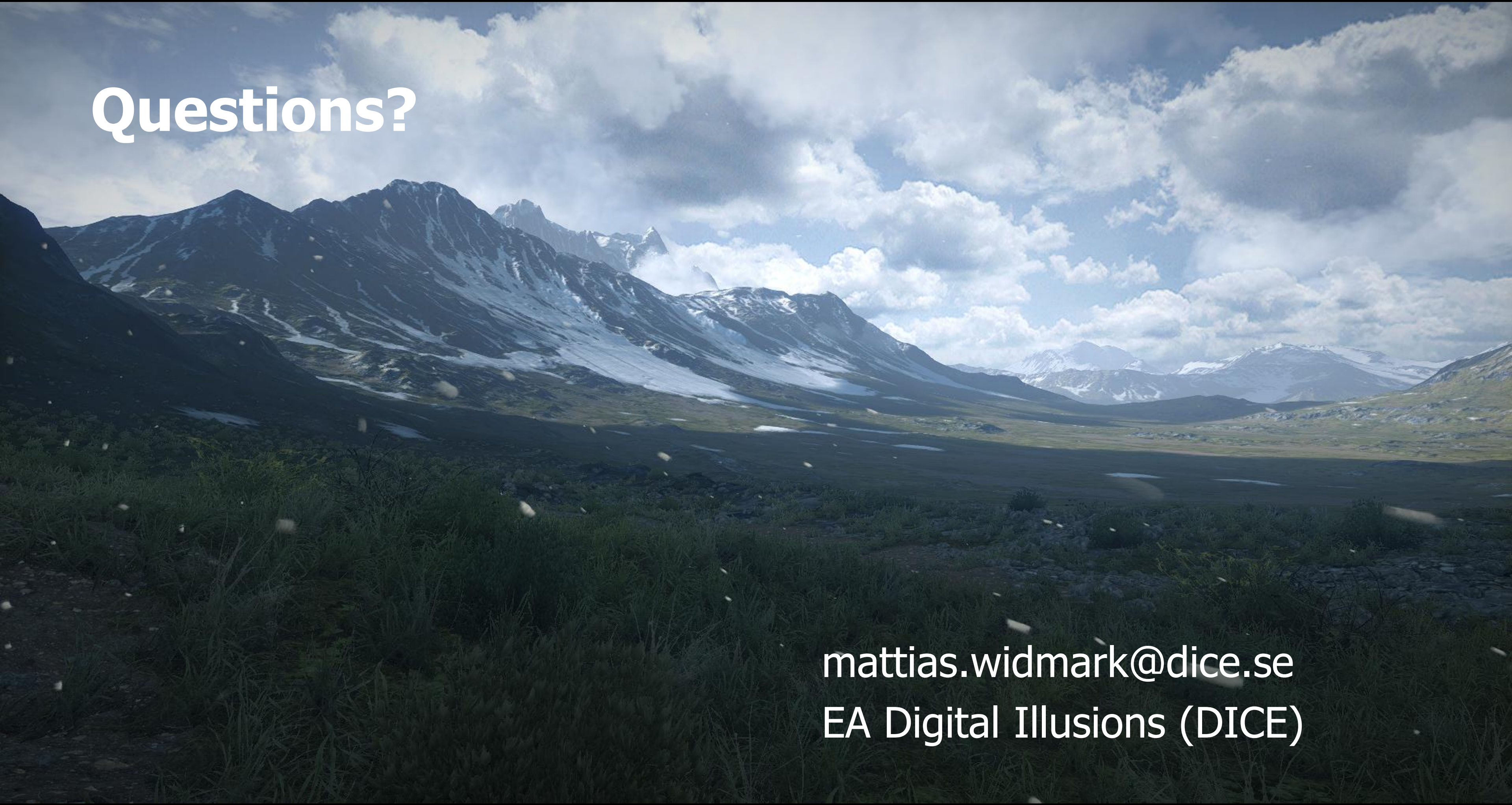
- Scalability – hierarchies, payloads and limitations
- Workflows – realtime in-game editing
- CPU and GPU performance
- Procedural virtual texturing – powerful GPU optimization
- Data streaming – minimizing memory footprint
- Robustness – global prioritization
- Procedural mesh generation
- **Conclusions**

Conclusions

- Frostbite 2 has a robust and competent terrain system
  - Heightfield, shading, decals, water, terrain decoration
- Most aspects scale well
  - View distance, data resolution, decoration density and distance
- Slick workflow
  - In-game editing
  - Good range of tools
- Good performance (CPU, GPU, memory)
  - Parallelized, streaming, procedural virtual texture

Special thanks

- The Frostbite team
- Black Box
  - Andy Routledge
  - Cody Ritchie
  - Brad Gour
- Criterion
  - Tad Swift
  - Matthew Jones
  - Richard Parr
- Dice
  - Andrew Hamilton



Questions?

mattias.widmark@dice.se
EA Digital Illusions (DICE)