



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

THE PREMIER CONFERENCE & EXHIBITION ON COMPUTER
GRAPHICS & INTERACTIVE TECHNIQUES

Large Scale Terrain Rendering in Call of Duty

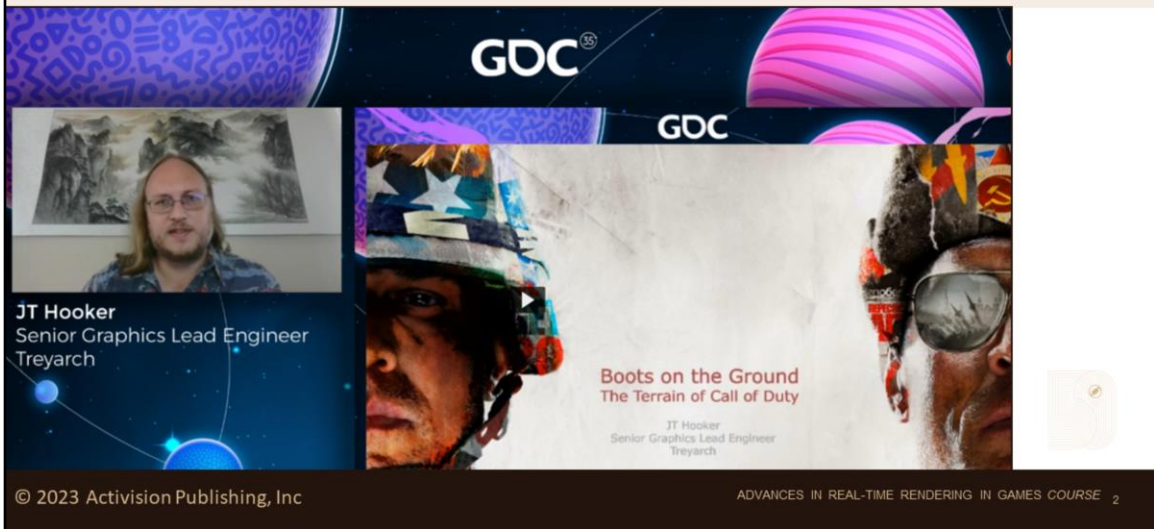
Advances in Real-Time Rendering in Games *course*

© 2023 SIGGRAPH. ALL RIGHTS RESERVED.

ADVANCES IN REAL-TIME RENDERING IN GAMES COURSE

- Hello, my name is Stephane Etienne. I work for High Moon Studios, an Activision owned studio in beautiful San Diego.
- I'll present the terrain technology that we use to render large scale terrains.

➔ Boots on the ground



- The terrain technology that is now used in Call of Duty was first implemented JT Hooker & crew from Treyarch many years ago.
- In his GDC 2021 presentation called “Boots on the Ground”, JT presented the super terrain technology at the time of Call of Duty Cold War.
- JT goes in a lot of details, about the authoring of super terrain. I’ll borrow some of his images in this presentation.
- However, I won’t go in as much details about several aspects of the super terrain technology, in particular anything that talks about authoring.

→ How Super Terrain works



A collection of surfaces

Concurrent development

Can't be scale nor rotated

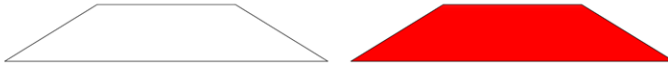
Must be square



1:00

- Super terrain is in effect a collection of terrain surfaces.
- In this screenshot, each purple rectangle is a terrain surface. Each red section is a point of interest, and it has its own terrain surface.
 - One reason we have so many terrain surfaces, is so that we can have concurrent development on the same map.
 - Another reason is so that we can varying amount of details.
- Terrain surfaces have several limitations.
 - They cannot be scaled nor rotated. They also must be square. It allows the code to be simpler and faster.
 - The terrains in red appear to be rotated and non square. This is achieved using cut out volumes, something I'll cover later.

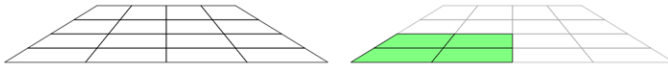
Depth 0
(Root)



Depth 1



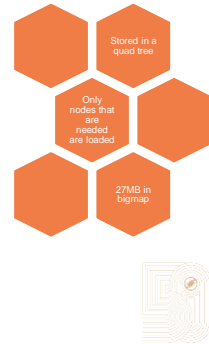
Depth 2



Depth 3



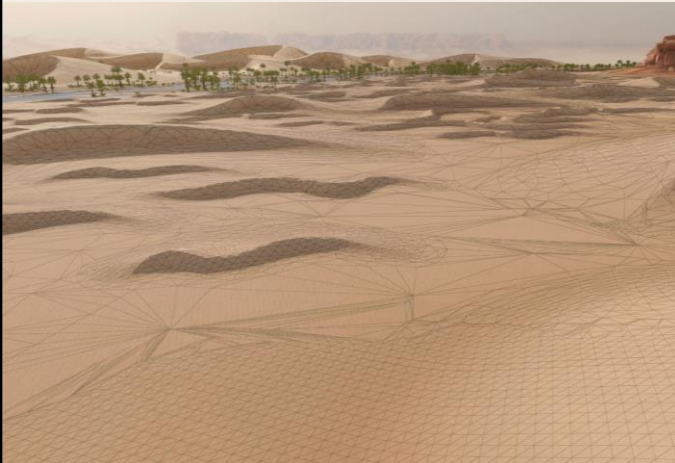
Depth 4



1:30

- The terrain technology makes heavy use of quad trees.
 - I'll assume you are all familiar with quad trees.
- Each terrain surface has a quad tree.
 - At each node, we store vertex & index buffers that must be loaded before the node can be rendered.
 - Nodes that don't need to be rendered, don't have any vertex or index data loaded.

→ Mesh Simplification



Simplified
meshes at
branches

- Constant vertex count at each branch
- Significant vertex count reduction
- Preserve silhouette
- Lerp towards low-rez vertex
- Only 8 bytes per vertex



2:30

- A terrain surface contains a height map and a quad tree.
- At each node of the quad tree, is a terrain patch.
- At the lowest level, the terrain patches use procedural vertices with all vertices placed on a regular grid.
- At the branches, we run our meshes through a mesh simplifier.
 - The mesh simplifier outputs 2 floats per vertex, an X & Y. The Z is reconstructed by sampling the height map.
- This allows us to retain significant amount of details with few vertices. We gain in visual quality but also in performance, because GPUs much prefer large triangles over small ones.
- Terrain patches also reference their lower resolution mesh
 - Vertices are interpolated towards their low-resolution counter part, as the camera moves away from the mesh.
 - When the camera is sufficiently far away from the mesh, and all vertices have collapsed to the lower resolution mesh, we switch to the lower resolution mesh.

→ Cut Out Map



Cut out volumes

- Make room for other terrain
- Make hole in the ground for caves
- Place buildings or other geo

Baked down to a cutout map

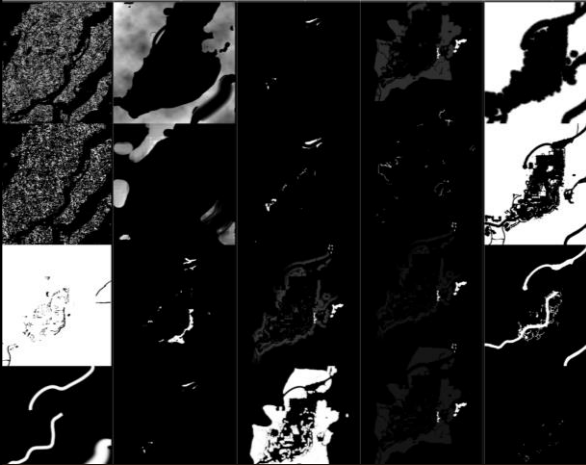
- 1 bit per pixel
- A byte encodes pixels in X & 2 pixels in Y



3:30

- When authoring terrain, artists will often use cutout volumes to mark area of the world where they don't want terrain.
- This is sometimes used when a low-resolution vista terrain needs to make room for another, more dense terrain, in a playable area.
- It is also used to make holes in the ground for caves, or to place buildings instead of terrain.
- Cutout volumes get baked down to one cutout texture map per terrain.
- The cutout texture map is very memory efficient. It requires only 1 bit per vertex.
- A single byte encodes the visibility state of 8 vertices, 2 rows of 4 vertices.

→ Material Layering



1 alpha mask per material

Lots of memory

Expensive & unpredictable
shaders

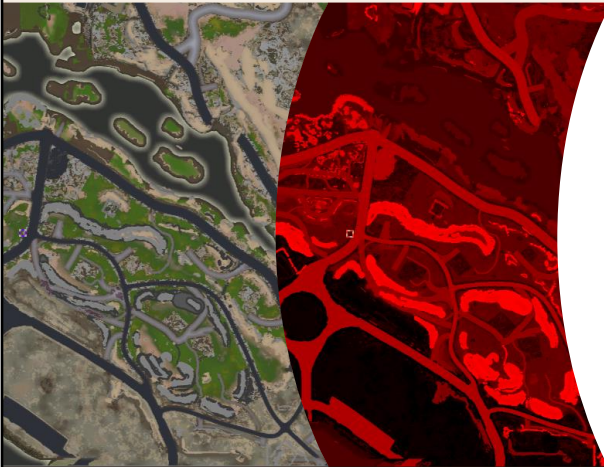
Only 32 layers



4:00

- A terrain surface is made of a collection of materials that we call terrain layers.
- Typical terrain layer would be grass, sand, dirt, etc.
- In our original implementation of our terrain technology, an alpha mask was used for each terrain layer.
- We only supported 32 terrain layers, but that was still a huge amount of memory due to all these alpha masks having to be loaded.
- It also implied the shader cost was unpredictable. Most pixels may have used only 1 layer, there was nothing to prevent all 32 layers from being used to shade a single pixel.
- Aside from that, 32 was too low a limit and we needed to increase that.

➔ One Material Per Vertex (OMPV)



1 Index map
per terrain

- 1 byte per texel
- Most prominent layer

1 Color map
per terrain

- BC1 compressed

Bounded
GPU cost

Up to 256
layers



© 2023 Activision Publishing, Inc

ADVANCES IN REAL-TIME RENDERING IN GAMES COURSE

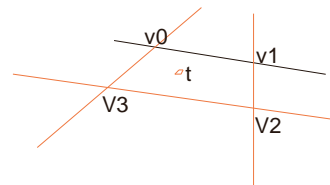
5:00

- In Black Ops Cold War, Treyarch replaced layer masks by a new technique they coined One Material Per Vertex, or OMPV for short.
- OMPV uses an index map, here shown in red, that holds a single byte per pixel and a color map, shown to the left of the index map, that holds a single RGB value per pixel.
- With OMPV, layer masks are gone, and since the color is BC1 compressed, this represent a significant memory saving.
- At each pixel, the index map records the most prominent layer. The color map is effectively a color tint that is meant to be used by that layer.

→ OMPV Explained



Compute	Gather	Build
Compute alpha of each vertex	Gather indices & colors at each vertex	Build unique list of indices <ul style="list-style-type: none">• Sum alphas & colors for duplicates• Multiply alphas by 2 to avoid issues with reveal maps• First layer has an alpha of 1



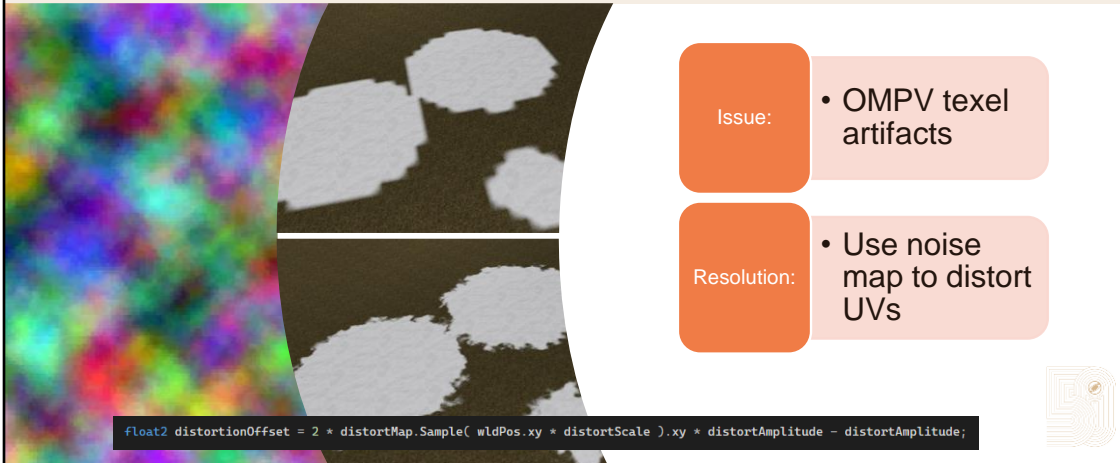
ADVANCES IN REAL-TIME RENDERING IN GAMES COURSE

5:30

- With OMPV, To shade the texel T here, the shader code gathers index & colors at the 4 surrounding vertices.
- We compute an alpha value that defines the contribution of each vertex.
 - The closer the texel T is to a vertex, the greater the alpha value for that vertex.
- The index of each vertex points to a terrain layer.
- Since each vertex is likely to reference the same layer, we can speed the processing of duplicate layers by summing up their alphas.
 - For example, if all vertices happen to be referencing the same layer, that's actually the common case, instead of sampling the same material 4 times, we will be sampling that material only once.
- Due to ill defined conditions that JT explained at length in his GDC presentation, we multiply the alpha by 2 and clamp to the 0 to1 range.
- We then blend all layers using the reveal maps that artists authored so that they have control over the blending process.
- Not all layers are the same. Layers are sorted by their index, the layer with the

lowest index is the primary layer and is never blended. It always has an alpha of 1.

→ Distortion



6:30

- OMPV has its fair share of issues though.
- The 1st issue has to do with the pixelated look that you get by implementing what I just described.
- It is clearly visible in the top right image here.
- To address that, we distort UVs using a global 256x256 distortion map depicted here.
- Artists can control the amplitude and frequency of the distortion on a per level basis.
- This is a map wide setting because UVs are distorted before index & color maps are sampled.

→ Tile Hiding

```
// Rotation is based on the world-space position of the texel, so it matches from terrain to terrain.  
// Any number that varies from texel to texel will do. Even if it repeats often it's ok, distortion hides this.  
// We reduce by 1/4 so we get sections of uniform rotation. Better for both visuals and performance.  
uint rotation = ( ( (int) ( ( floorCoord.x + offsX[i] ) * 0.25 ) % 2 ) + 3 * ( ( (int) ( ( floorCoord.y + offsY[i] ) * 0.25 ) % 2 ) ) % 4 ) & 0xF;
```

Pick random
rotation per
texel

Based off
world
position

16 unique
rotations

Treated as
radians

Increased
shader cost

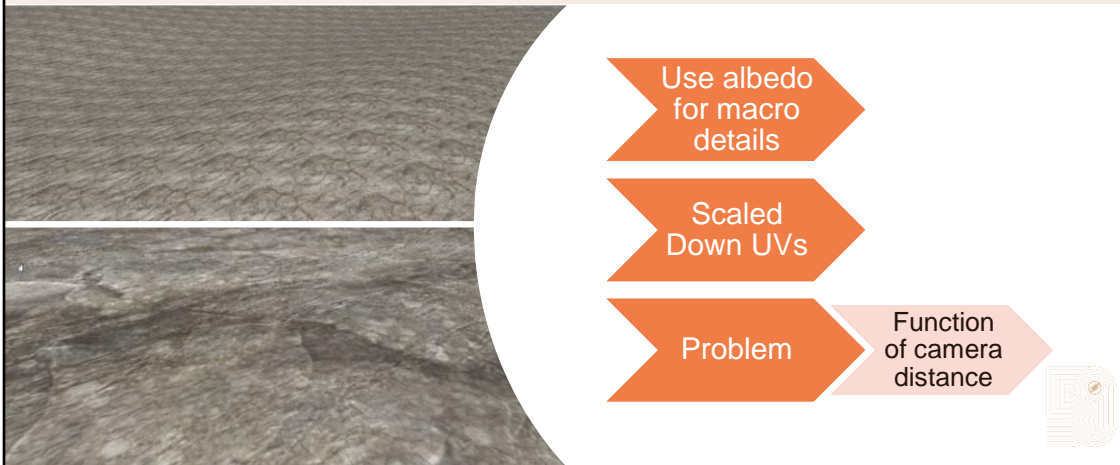
Can't merge
layers
anymore



7:00

- Not unique to OMPV, but when a terrain layer is used on large swath of the map, tiling artifacts become noticeable.
- The way we address that is with a technique that we coined tile hiding.
- At every vertex, we compute a random rotation based on the world position of that vertex.
- This rotation is an integer, in the range 0 to 15, interpreted has an angle in radians.
- Tile hiding is very effective at hiding tiling artifacts, but due to this added rotation, we can't merge duplicate terrain layers anymore, unless they happen to have the same random rotation.
 - In the picture on top, since most vertices would reference the grass material, shading a pixel would typically involve sampling the grass material once.
 - After tile hiding though, since the 4 vertices are likely to have different rotations, shading a pixel will require sampling the grass material 4 times, each time using different Uvs.
 - Still the visual quality improvements are well worth it.

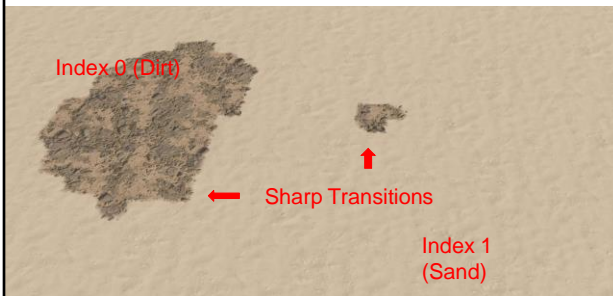
→ Vista UVs



8:30

- Another issue with using the same texture on large swath of the map, is eventually all details from the highest mip collapse to a single color when seen from far away.
- A solution to that problem is called Vista Uvs.
- Using Vista Uvs, we compute a second set of Uvs, that are quite simply the original set of UVs, except scaled down, to magnifying the texture being sampled. We sample albedo & normal again to compute the macro contribution.
- We lerp between regular contribution and macro contribution using camera distance to compute final contribution.
- On some type of content, such as rocks, Vista Uvs does a good job at creating interesting details when viewing the content at a distance.

→ OMPV Problems



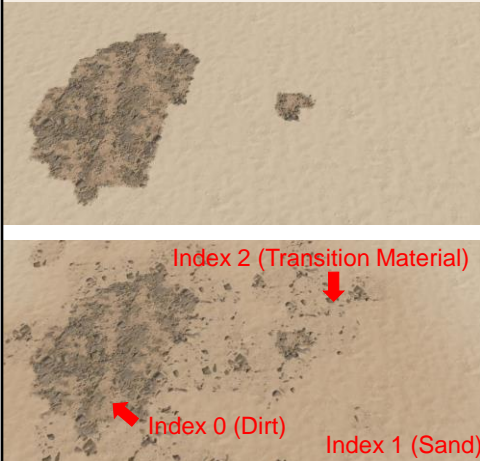
Sharp transitions



9:30

- Another issue with OMPV is the sharp transition when going from 1 material to another.
- Here, we show what happens when going from dirt to sand.

→ Content Work-Around



Solution:

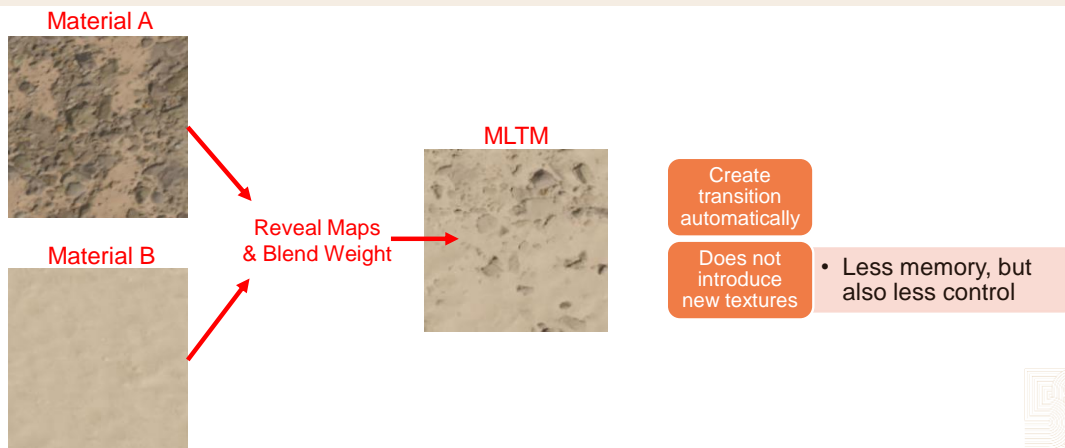
- Manually authored transitions



9:30

- The solution we have been using so far is to have artists manually author transition materials.
- It works well, but it is a manually intensive process.
- Since each transition is a brand-new set of textures, it adds pressure to our texture streaming system and consumes precious memory.

→ Multi-Layered Terrain Materials (MLTM)



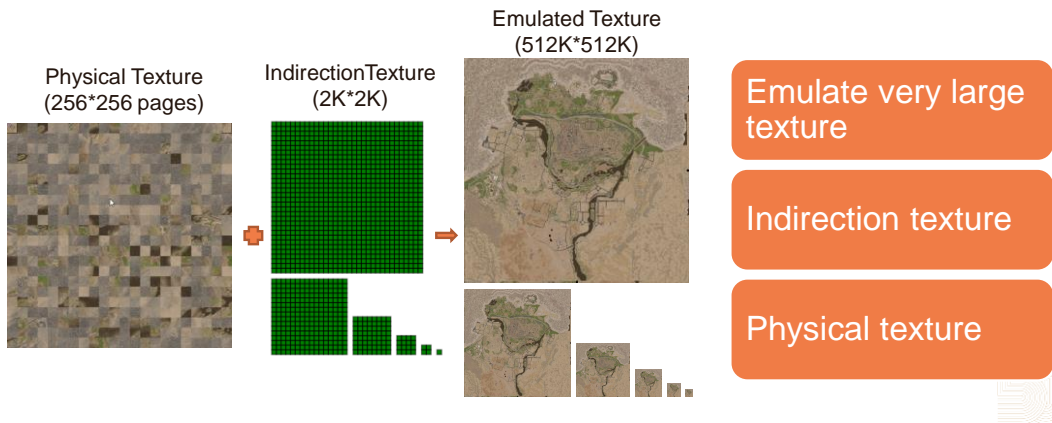
© 2023 Activision Publishing, Inc

ADVANCES IN REAL-TIME RENDERING IN GAMES COURSE

10:00

- We are currently implementing a solution to that problem, we call Multi-Layered Terrain Materials or MLTM for short.
- A Multi-layered terrain material takes 2 input terrain materials, and a threshold that is used to blend the 2 materials. The reveal map of the target material is used to complete the blend.
- In essence, this is like having transitions authored in Photoshop, but without the memory cost.
- All this is done at runtime. This does increase the cost of the shaders, but since transition materials are only used during transitions, this is ok.

→ Virtual Texture



10:30


- Virtual Texturing is a process by which a very large texture is emulated at runtime.
- Similar to virtual memory emulating a computer with significantly more physical memory than it has, virtual texturing can be used to emulate a texture much larger than would otherwise be possible
- The keys to that emulation are 2 textures:
 - First, there is a physical texture that holds pages of the very large textures.
 - A page is a small square section of the emulated texture or one of its mips
 - A page is typically 256x256 pixels, but it can be any size you want
 - 2nd, there is an indirection texture that holds enough information to redirect the GPU to sample texture data stored in the physical texture.
 - A single pixel of the indirection texture will point to a page in the physical texture.
- For example, a 4K*4K physical texture of 256x256 pages times a 2K*2K indirection texture can be used to emulate a 512K*512K texture.
- Virtual texture is often called procedural virtual texture because the emulated texture is the result of procedural composition of materials and other drawing


primitives.

- In this presentation, procedural is omitted but it is important to remember that the emulated texture is a result of a complex blending process and does not exist on disk.

→ Adaptive Virtual Texture (AVT)




 Emulate a texture that can have up to 24 mips

 25 pixels/inch (10 pixels/cm)

 Max terrain size: 10.6*10.6 miles (16.8*16.8km)

 World is sliced in 210*210 feet sectors (64*64 meter)

 Each sector is managed by a virtual image

 Size of virtual image is controlled by camera distance



© 2023 Activision Publishing, Inc

ADVANCES IN REAL-TIME RENDERING IN GAMES COURSE

- 12:00
- Virtual textures are great at emulating large textures that cannot fit on the GPU due to their size.
- However, they are still too small to emulate humongous textures.
- For example, the 512K by 512K virtual texture in the previous slide, would only cover about .3 by .3 miles at the resolution of 25 pixels per inch.
- For much larger virtual textures, Ka Chen from Ubisoft proposed Adaptive Virtual Texturing in his GDC 2015 presentation.
- I'll shorten Adaptive Virtual Texturing to AVT from now on.
- AVT can emulate a texture with up to 24 mips.
- At a resolution of 25 pixels per inch, this is enough to cover a 10.6 by 10.6 miles area.
- At the core, the world is sliced in sectors that are about 210 by 210 feet across.
- Each sector is managed by a virtual image, which is in essence a regular virtual texture.
- The size of that virtual image is dynamic. It changes as a function of the camera distance. Hence why it is called Adaptive Virtual Texturing.

→ Virtual Image



At most 16 mips

Only sectors around camera are active

Default sector covers entire world

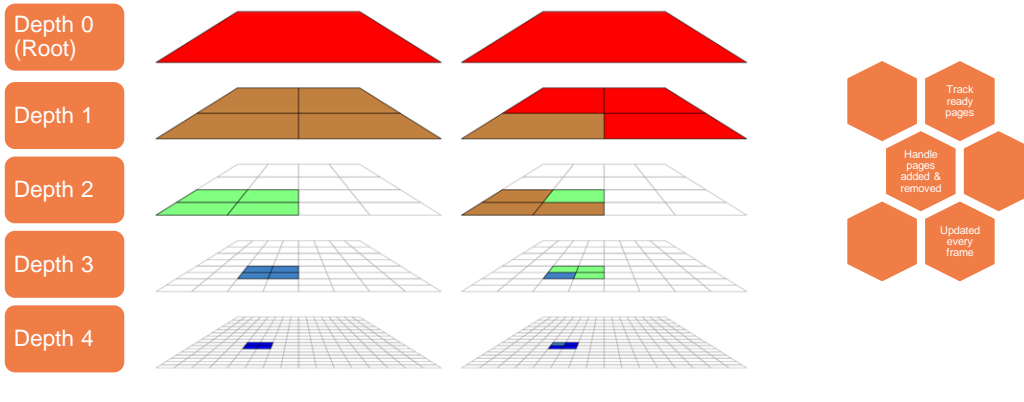


13:00

- A virtual image has at most 16 mips.
- With 16 mips, we can cover an area of 210 by 210 feet with a resolution of 25 pixels per inch
- Only sectors around the camera are made active. We only support up to 255 sectors.
- There is also a default sector that covers the entire world.
- The default sector always uses a 16 mips virtual image.
- >>>>>>
- In this video, you can see virtual images being created.
 - The square in the top left of the screen is a representation of the indirection texture. It is only 512 by 512 pixels across.
 - The default sector in pale green occupies the top left corner.
 - The virtual images allocate space for themselves out of that indirection texture.
 - The same color scheme is used to shade the terrain so that you can see the regions of the terrain that are under the control of which sector.

- As we get close to a sector, you can see its virtual image getting bigger. As we move away from a sector, its virtual image shrinks.

→ Virtual Image Quad Tree

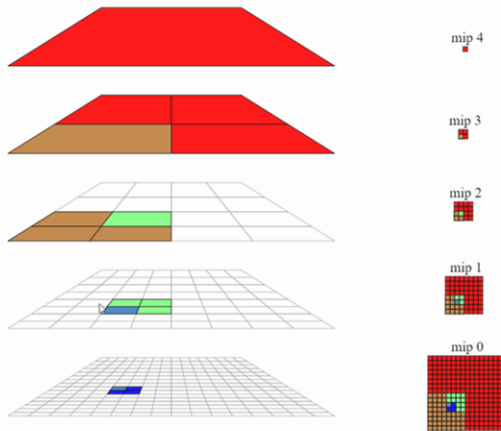


14:00

- A virtual image quad tree is more complicated than normal quad tree because it tracks pages that are usable by the GPU.
- When expanding a branch of the quad tree, some child nodes may not have their own page. As a result, they must point to the parent page.
- This is illustrated in this picture.
 - For example, at depth 1, the children of the root page were expanded because the child in brown had to be created, given there was a page for it. The other 3 children simply reference the parent page.
- Every frame, pages may become ready for the GPU to use, or the opposite, pages may become unavailable.
- When a new page is composited, the quad tree isn't updated until the next frame. This is because we cannot take the risk of telling the GPU to start using a page as it is being composited.
- When a page is not requested, it will age in the cache. It is removed from the quad tree but we guarantee the page won't be recycled for at least 3 frames. This is more than enough to ensure that the GPU does not use a page that is being

recycled.

→ Indirection Texture



Copy of quad tree for GPU

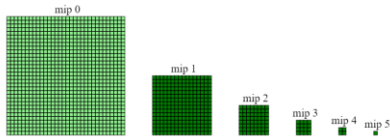
Updated by compute shader



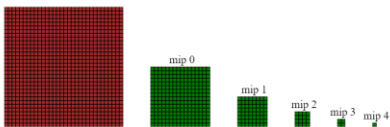
15:30

- The CPU version of the quad tree is too complex for the GPU to consume as is.
- So instead, we bake the quad tree in an indirection texture.
- Every frame, the game computes the delta between the previous frame quad tree and the current frame quad tree.
- This gets translated into commands that are then processed by a compute shader, and it is that compute shader that updates the indirection texture.
- The picture here shows what the CPU quad trees look like from the perspective of the GPU.

→ Dynamic Resizing



New mips added when camera gets closer



Mips removed when camera moves farther



16:00

- When the camera moves closer to a sector, the game may decide a new mip needs to be added.
 - It first allocates a bigger sub indirection texture from the main indirection texture.
 - Then it populates it, using the same process used to update the indirection texture frame over frame.
- When the camera moves further away, the reverse process takes place.
 - That is mips are removed from the indirection texture.

→ Indirection Texture Texel

Phys Page X : 8	Phys Page Y : 8	Mip Level : 8	Quad tree width : 8
-----------------	-----------------	---------------	---------------------

Coordinates of physical page

Mip level of physical page

Depth of quad tree

```
Float2 SectorUVsToPageUVs( float2 sectorUVs, uint quadTreeWidth )
{
    const float2 indirectionTexelCoordAtMip = sectorUVs * quadTreeWidth;
    const float2 indirectionTexelIndexAtMip = floor( indirectionTexelCoordAtMip );
    const float2 pageUVs = indirectionTexelCoordAtMip - indirectionTexelIndexAtMip;
    return pageUVs;
}
```



16:00

- An indirection texel is a 32 bit value that provides the coordinate of the physical page.
- It also specifies the mip level of the page and the quad tree width.
- All this information is required to go from sector Uvs to physical page Uvs.
- The code snippet here shows how we convert sector Uvs to physical page Uvs.

→ Indirection Texture In Action



- 16:30
- >>>>>>>
- We have tools to help debug and visualize the indirection texture in-game.
- This is what it looks like.
- Each square on the terrain is just the 32 bits indirection pixel interpreted as an albedo color.
- Normally, these would be filled by a physical page.

→ Terrain Rendering



Full Screen Quad

Terrain Stencil Bit



17:00

- Terrain is rendered both during the prepass and the opaque pass.
 - During the prepass, terrain writes to the depth buffer and to a stencil bit reserved for super terrain. We also write the geometric normal to a g-buffer.
 - During the opaque pass, terrain is deferred rendered using a full screen quad. Every pixels that have the stencil bit set, sample from the virtual texture to shade the current pixel.

→ Feedback Buffer



240x136

Populated by the GPU
during the opaque

Write a 32 bits value

No thread
synchronization



17:30

- >>>>>>>
- A core aspect of virtual texturing is that it is driven by the GPU.
- During the opaque pass, as it shades every pixel, it also computes the physical page it would have liked to use if it had been available.
- This information is stored in a feedback buffer
- The feedback buffer is read back by the CPU 3 frames later..

- Its resolution is 240 by 136 pixels across but we are investigating smaller resolution feedback buffer on low-end hardware.
- For performance reasons, there is no thread synchronization.

→ Feedback Buffer Value

Mip Level : 8	Sector ID : 8	Quad Tree Y : 8	Quad Tree X : 8
---------------	---------------	-----------------	-----------------

```
uint FeedbackBuffer_ComputePixelValue( float2 sectorUVs, float virtualImageWidth, float mipBias, uint sectorID )
{
    const float2 texcoords = sectorUVs * virtualImageWidth;
    const float2 dx = ddx( texcoords );
    const float2 dy = ddy( texcoords );

    const float px = dot( dx, dx );
    const float py = dot( dy, dy );
    const float maxLod = 0.5 * log2( max( px, py ) );
    const float minLod = 0.5 * log2( min( px, py ) );
    const float mipLevel = lerp( maxLod, minLod, mipBias );

    const int iMipLevel = int( floor( mipLevel ) );

    uint quadTreeXY = uint2( floor( sectorUVs * 256.0f ) );
    quadTreeXY >>= iMipLevel;

    return ( uint( iMipLevel ) << 24 ) | ( sectorID << 16 ) | ( quadTreeXY.y << 8 ) | quadTreeXY.x;
}
```











18:00

- Each terrain pixel writes a 32 bit value into the feedback buffer during the opaque pass.
- It writes the mip level of the virtual texture it would have liked to use, the sector ID of the pixel and the quad tree coordinates of the indirection texture quad tree node it would have like to use.
- The most difficult part of the feedback buffer is the mip level. It is computed by looking at the UV gradients. The code snippet above illustrates what we do.

→ Processing of the Feedback Buffer



-  Count all unique values
-  Children count added to parents
-  Sort by popularity
-  Bias sector root pages
-  Use camera velocity to bias high-rez pages
-  Pages above threshold are selected
-  Pages already visible are rescued
-  Never request more than 1/3rd of the total pages



18:30

- Once the GPU is done writing to the feedback buffer, the CPU reads it back.
- Given the resolution of the feedback buffer is 240 by 136, it means we process about 32000 values.
- First, we count all unique values.
- Then, we propagate children counts to their parent. This is to ensure that if children don't have enough hits, hopefully the parents will.
- Then we sort by popularity. We want the pages that are requested the most to be composited first.
- We bias root pages of the sectors, so that if a sector is at all visible, there is at least 1 page for that one sector.
- We also bias against high-resolution pages as a function of the camera velocity.
- Given the feedback buffer is updated with unsynchronized threads, it can be noisy. We address the issue by using a threshold below which we don't bother

compositing the page.

- Sometimes, pages that are close to the threshold may be above the threshold one frame, and below it the next. So, we rescue pages that are already visible to avoid visual artifacts.
- We also never request more than $1/3^{\text{rd}}$ of the total number of pages in the cache to avoid churn.

→ Battle Against Low-Rez VT

Fake pages

- Hidden sectors
 - At least a page 1/16th of virtual image
- Camera teleports
 - Create pages where camera will be

Composite even when not ready

- Missing mips better than low-rez VT
- Ascending waits



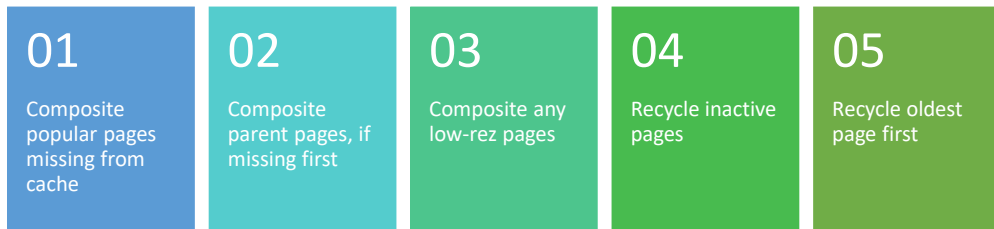
19:30

- Despite everything I described, it is still possible to see low resolution VT from time to time.
- For example, if you face the camera in one direction and turn it to the side so that it fills the cache with content requested on screen, if the camera then do a quick 180 degree rotation to look what is behind it, it will take a few frames for new pages to be added to the cache. This can be quite noticeable.
- The solution is to pick the closest medium resolution page of each sector and add it to the list of requested pages.
- We often do camera teleport, or very quick camera transitions, for example transitioning from the sky where players pick their load-out to the ground where the player actually spawns. The game is always told where the camera will teleport so that we can stream the appropriate content for the target location.
 - AVT piggybacks on that to create medium resolution pages as well.
- Virtual texturing is essentially a cache. Typically, you only want to populate caches with up-to-date data. If the VT wants to composite a page, we need to make sure

all the mips that are needed for that page are loaded before the page is composited.

- Unfortunately, on hardware that have slow storage devices, it could take several seconds to load the required mips.
- This can result in bugs where the camera effectively outruns the VT and the entire terrain looks like a PS1 game.
- We found that it is better to composite the page as soon as it is requested, but mark it low-rez.
- 15 frames later, if it still isn't fully ready, we recomposite it again, assuming we have bandwidth for it.
- That delay is doubled to 30 frames, and we recomposite it yet again.
- We do that until the page is marked as ready and we composite it a final time.

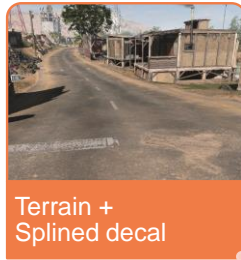
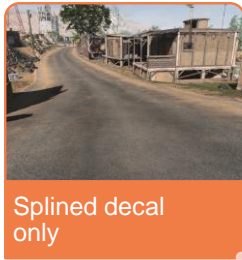
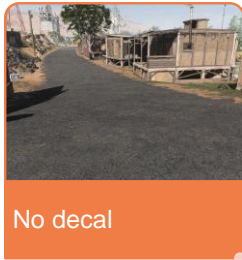
→ Composition Manager



21:30

- Every frame, we only composite a finite number of pages. On low-end mobile hardware, that never exceeds 4 pages but on high-end PCs, this goes up to 20 pages per frame.
- Popular pages, not in the cache, are composited first.
- If we have space left, we recomposite any low-rez pages we may have. Low-rez pages are priority sorted, so we recomposite these pages that need it the most first.
- All pages have an age. Pages that are requested have their age reset to 0. Pages that have not been requested see their age incremented.
- Pages that have not been requested for at least 3 frames can be recycled.
- We always recycle the oldest page first.




→ Decal Rendering



22:00

- Decals play a critical role when rendering super terrain.
- We have 3 kinds of decals.
 - Terrain decals are top-down rectangular orthographic projections.
 - Splined decals are top-down orthographic projections but follow a Bezier curve.
 - Volume decals are not limited to top-down projection. They also work on any kind of geo, including terrain.
- First screenshot shows what the world looks like with just terrain layers, and no decal.
- In the 2nd screenshot, terrain layers & terrain decals are rendered.
- In the 3rd screenshot, terrain layers & splined decals are rendered. As you can see, splined decals is what we use to make roads.
- The final screenshot shows what it looks like when everything comes together.

→ Streamer Connection

-  Use camera distance to pre-stream mips
-  Ensure decal tree nodes are loaded
-  Given a page world bounds, validate required mips are loaded
-  If all mips are ready, mark page as ready
-  If some mips are missing, mark page as low-rez
-  If critical data is missing, mark page as not ready

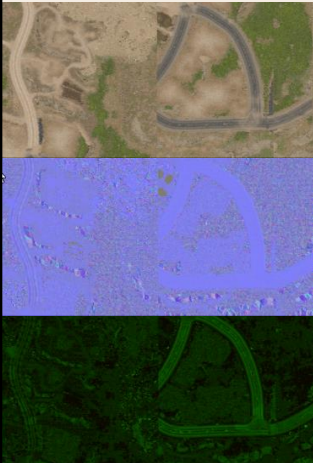


23:00

- We call streamer the game system that is responsible for loading assets from disk, that is usually dependent on camera location.
 - For example, the streamer is responsible for loading image mips required to render the scene at full resolution.
 - Streamer will also load mesh level of details, sounds, animations and various other kind of data.
- Super terrain uses a lot of data that is streamed from disk.
 - Mesh data such as vertex & index buffers are streamed as a function of camera distance.
 - Decals are stored in a quad tree and this is also streamed.
 - Given the current camera location, the game computes the image mips for terrain layers and decals that ought to be loaded.
 - When a request comes in to composite a page, the game checks that the appropriate mips are loaded.
 - If they are not loaded, we increase their priority, so that they get streamed as quickly as possible.
 - If they are any missing mips, we still composite the page but mark it as low-

rez.

- If critical data is missing, such as missing decal tree nodes, we mark that page as not ready.
 - Rendering a page with missing drawing primitives, would result in popping artifacts.
- If nothing is missing, we mark the page as ready.



Writes to
3 scratch
buffers

- Albedo
- Normal
- Metalness, Occlusion & Gloss
- Uncompressed



24:00

- Terrain composition uses 3 G-buffers called scratch buffers.
 - The 1st one holds the albedo.
 - The 2nd one holds the normal.
 - While the 3rd holds one metalness, occlusion & gloss.
- When compositing a page, we write to all 3 buffers at once.
- Scratch buffers are uncompressed render targets, and as such can be fairly expensive in terms of memory use.

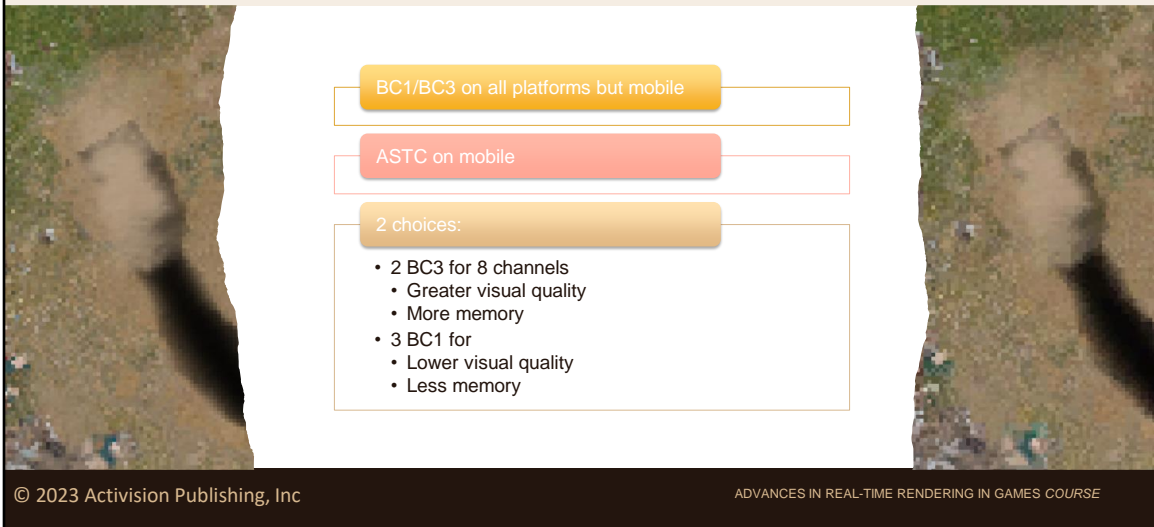
→ Composition



24:30

- We first render terrain layers using a compute shader.
 - Followed by splined decals using a vertex & pixel shader.
 - Followed by terrain decals using a vertex & pixel shader as well.
 - And we finish with the volume decals, that are rendered using a compute shader.
- Decals can be sorted between each other, but it is not possible for say a volume decal to render before a terrain decal.

→ Compression



BC1/BC3 on all platforms but mobile

ASTC on mobile

2 choices:

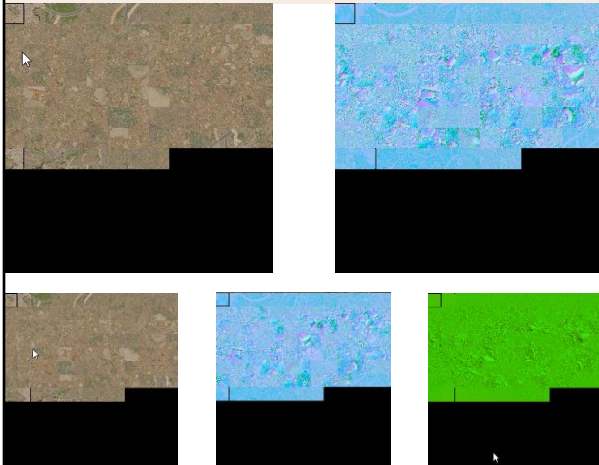
- 2 BC3 for 8 channels
 - Greater visual quality
 - More memory
- 3 BC1 for
 - Lower visual quality
 - Less memory

© 2023 Activision Publishing, Inc

ADVANCES IN REAL-TIME RENDERING IN GAMES COURSE

- 25:00
- Before copying the composited scratch pages to the physical texture, we need to compress them.
- This is done using compute shaders that we wrote.
- On mobile, we use ASTC compression and BC compression on all other platforms.
- On all platforms but mobile, data can either be compressed as 2 BC3 textures or 3 BC1 textures.
- Note that this is at this stage that we see the greatest loss in visual quality.

→ Physical Textures



2 Mips

4 pixels border

- True page size: 248x248 pixel

BC3

- Albedo+Metalness
- Normal+Occlusion+Gloss

BC1

- Albedo
- Gloss+NormalX+Occlusion
- Metalness+NormalY+Feature

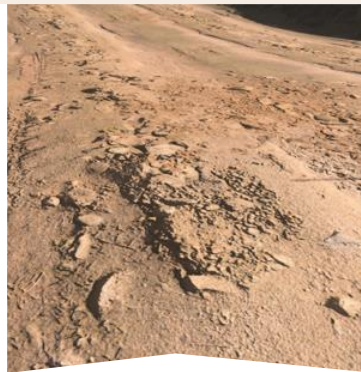
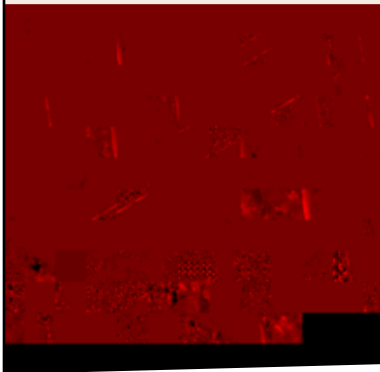


25:30

- Our physical textures have 2 mips. This is required for bilinear texture interpolation.
- Our physical pages have 4 pixel borders. This is required for 8X anisotropic filtering to work.
 - Note that this means that we are only using 248 by 248 pixels per page.
- Our BC3 physical textures have 8 channels to play with.
 - The first texture encodes albedo and metalness, while the 2nd encodes normal, occlusion & gloss. The X component goes in the green channel while the Y component goes in the alpha channel.
- In the BC1 mode, we have 9 channels to play with.
 - The first texture encodes the albedo.
 - The 2nd texture encodes the gloss, the x component of the normal and the occlusion.
 - While the 3rd texture encodes the metalness and the Y component of the normal. We have 1 channel left for anything that we want.
 - Normal components are encoded in the green channel as the green channel has 6 bits instead of 5 for the other channels.

- Although in theory the 2 BC3 mode should result in better visual quality, I have never been able to tell the difference.
- The BC1 mode reduces memory by a 3rd and adds 1 channel, which is nice.

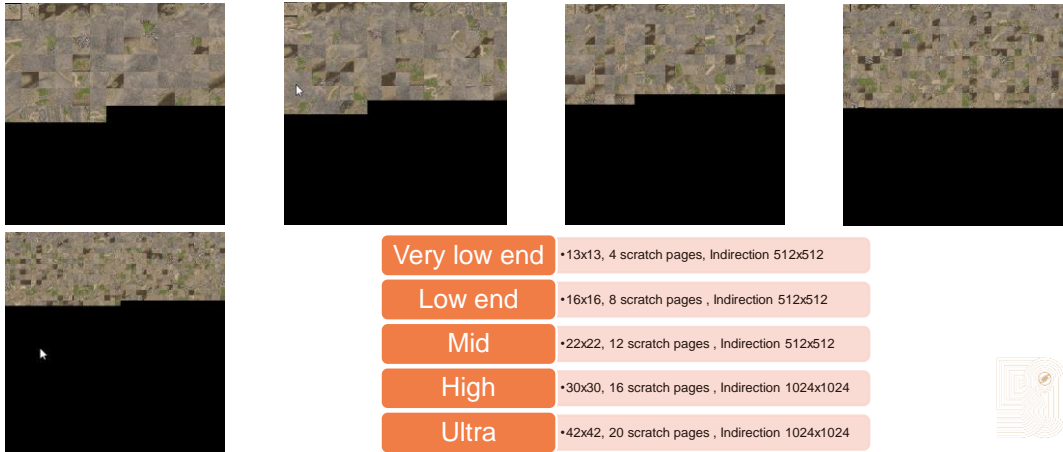
→ Displacement



26:30

- On platforms that have GPU Tessellation & Displacement enabled, we also use virtual texturing to store displacements.
- A displacement physical page is 32 by 32, has 1 mip, and has 1 pixel border, because it is only used to displace vertices, and the density of these vertices is significantly less than pixel density.
- Additionally, it is BC4 compressed, so memory cost is quite small.
- Note that displacements don't affect collision, which means they cannot be too big, otherwise this will start causing noticeable issues.

→ Scalability



27:00

- Because our terrain technology runs on a wide range of devices, AVT can be tuned to fit the platform:
- We have 5 quality settings:
- In Very low setting, the physical texture is only 13 by 13 pages. Indirection texture is 512 by 512
- In low setting, the physical texture is 16 by 16 pages. Indirection texture is also 512 by 512.
- In mid setting, the physical texture is 22 by 22 pages. Indirection texture is still 512 by 512
- In high setting, the physical texture is 30 by 30 pages. Indirection texture is 1024 by 1024.
- In Ultra setting, the physical texture is 42 by 42 pages. Indirection texture is 1024 by 1024.

➔ Offline Baked Virtual Texture



Root page persistent

Reduces GPU load

Reduces streamer pressure

Used during lighting bake & proxy lod generation



28:30

- VT pages for the top 6 levels of the mip chain are baked offline. Usually, these pages are quite expensive as they cover large swath of the map.
- By baking them offline, we significantly reduce GPU pressure, and we can really crank up the visual quality as well.
- Here, you can see the world rendered using only the offline baked pages.
- It is obviously too low-rez for anything that is close to the camera, but at a distance, offline baked pages is what the GPU would want to use anyway.
- Although these images have to be streamed, it ends up saving memory, because we don't need to stream terrain layers & decals for terrain surfaces that are sufficiently far away. This saves memory and reduces streamer pressure.

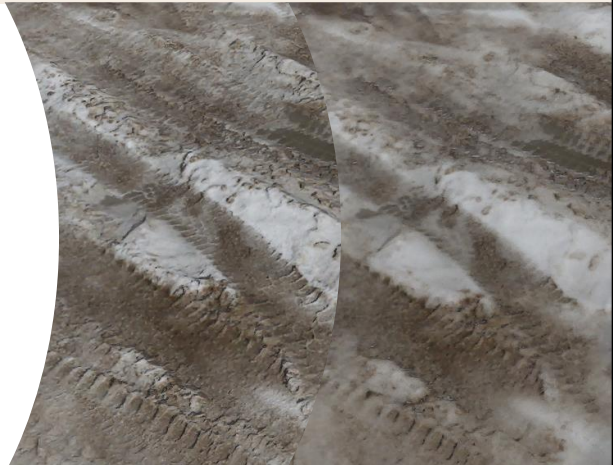
→ Extra Channel

Scattering

Thermal

Emissive

Glint



- 29:30
- When using the 3 BC1 mode, 9 channels are available. Since AVT normally only uses 8 channels, it means we have 1 channel left to use for something.
- So far, we have come up with 4 different uses for this channel.
 - Scattering is often use on snow to make the snow look more like the real thing, as can be see from this picture.
 - Thermal is use for night and heat vision.
 - With emissive, you can place ambers on the ground.
 - Glint is used on sand or snow to make them sparkle.
- Enabling a feature is done at the map level. We can only support one feature at any one time.

→ Unified Height Map



Single Height
Map

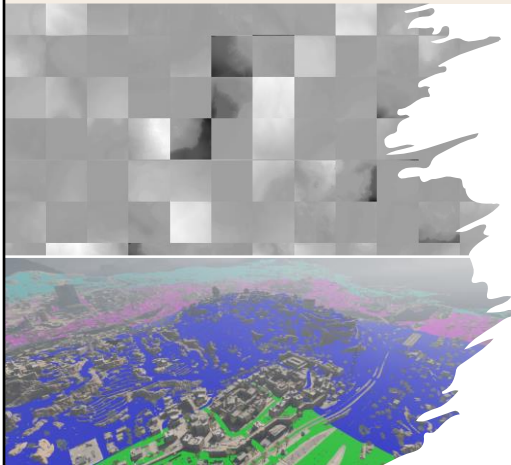
Must have for
Skirts



30:00

- When each terrain surface has its own heightmap, knowing the height of any point on the super terrain is really hard to do.
- That becomes trivial if a single height map is used to cover the entire super terrain.
- We call this unified height map or UHM for short, because all the heightmaps are merged into just one map.
- With the UHM, it becomes possible to implement skirts, something artists had long been asking for.

→ Virtual Height Map



Virtualized Unified Height Map

Quad tree

64x64 pages

2 pixels border

144 pages

32MB -> 1.2MB

8KB indirection table



30:00

- A typical UHM will consume 32MB of memory, but artists would be happier if it was 128MB or more as it means it has greater definition.
- 32MB is a lot of memory on some platforms.
- By virtualize the UHM, we significantly reduce its memory footprint.
- We call that Virtual Height Map or VHM for short.

- The VHM has 144 pages, and each page is 64x64, not including a 2 pixels border.
- This means the cache is a little less than 1.3MB.
- It also uses an indirection table, similar to the indirection texture that VT uses, except it is just a buffer. For our largest map, it is only 8KB in size.
- Pages from the VHM are prioritized by their distance to the camera.

- We always fill the cache.
- Pages that are in the cache have a hysteresis to make it harder for them to be ejected.
- In this rendering of the map, you can see the resolution of the terrain tiles. Green tiles are at the highest resolution, followed by blue, purple then cyan.

→ Height Map Upsampling



Goal

Reduce disk footprint
Potentially not stream at all



Process

Predictor guess child height
Error stream provides delta
Only error stream needs to be serialized

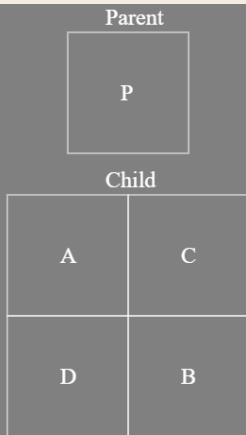


© 2023 Activision Publishing, Inc

ADVANCES IN REAL-TIME RENDERING IN GAMES COURSE

- 31:30
- Early on, we evaluated multiple upsampling techniques, in the hope that maybe we could keep the entire VHM loaded in memory but compressed.
- It turns out we still had to stream, but this is interesting research nonetheless.
- All this research was done using a heightmap downloaded from the United State Geological Survey website.

→ C0 Upsampling



Results:

- 60% of source

Details:

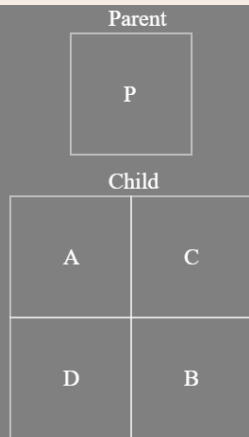
- $A = B = C = P$
- $D = 4 * P - (A + B + C + 2)$



32:00

- In this slide and the next ones, P is a parent texel, and code needs to predict the height at the 4 child texels.
- P is the average of the height at A, B, C & D.
- We always reconstruct all the As, followed by the Bs, Cs then Ds.
- Simplest image upsampling technique is to assume that child height is the same as parent height. This is our C0 predictor.
- We can do something a little more complicated for D, now that we know the value of A, B & C.
- This simple upsampling technique reduces memory to 60% of the source.

→ C0 Upsampling plus



Results:

- 59% of source

Details:

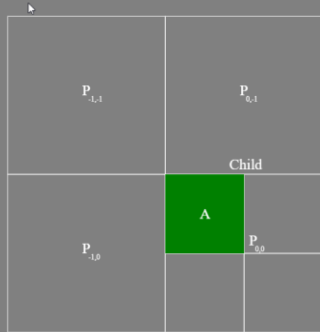
- $A = P$
- $B = (P * 4 - A) / 3$
- $C = (P * 4 - (A + B)) / 2$
- $D = 4 * P - (A + B + C + 2)$



32:30

- C0 upsampling plus improves on the previous technique by making use of the As to reconstruct the Bs, then As & Bs to reconstruct the Cs.
- We only gain a paltry 1% by doing that though.

→ C1 Upsampling: A



Results:

43% of source

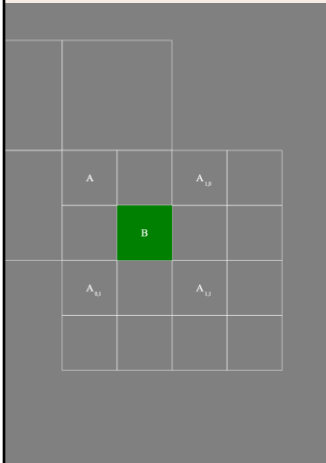
A:

$$(P_{-1,0} + P_{0,-1} + P_{-1,-1} + 9 \cdot P_{0,0}) / 12$$



- C1 up sampling makes use of neighbors and reduces memory use to 43% of the source data.
- Here is how we reconstruct the As.

→ C1 Upsampling: B



Results:

43% of source

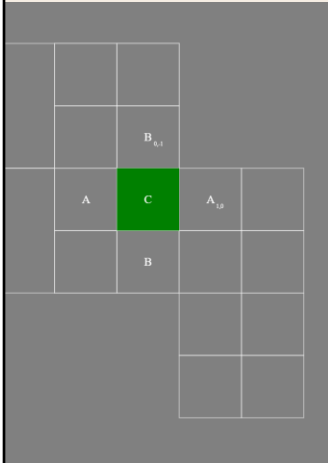
B:

$$(A + A_{1,1} + A_{0,1} + A_{1,0}) / 4$$



- For the Bs, we simply average the neighboring As

→ C1 Upsampling: C



Results:

43% of source

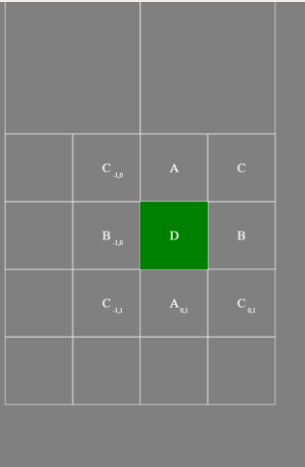
C:

$$(A + A_{1,0} + B_{0,1} + B) / 4$$



For the Cs, we average the neighboring As & Bs

→ C1 Upsampling: D



Results:

43% of source

D:

$$(A + A_{0,1} + B_{-1,0} + B + C_{-1,1} + C + C_{-1,0} + C_{0,1}) / 8$$



For the Ds, we average the As, Bs & Cs. Because the Cs are further away from the D texel, we ought to use a lower contribution weight, but our implementation does not do that right now.

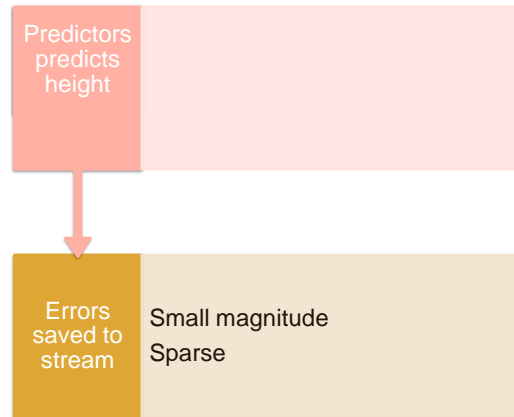
→ C2 Predictors

Tangent:	Bezier:	Neville:	Neville Incremental:	5x5 Kernel:
<ul style="list-style-type: none">• USGS: 42%• Game: 42%	<ul style="list-style-type: none">• USGS: 39%• Game: 42%	<ul style="list-style-type: none">• USGS: 33%• Game: 41%	<ul style="list-style-type: none">• USGS: 32%• Game: 40%	<ul style="list-style-type: none">• USGS: <32%• Game: N/A



- 33:30
- We also implemented C2 predictors, that are essentially like C1 predictors, except they look at further neighbors, so as to reconstruct a C2 continuous curve, instead of just a straight line.
- These worked quite well on the USGS map, but didn't provide as many gains with the height maps that we use in game.
- We also evaluated a simple neural network, and although it did better than our best C2 predictor, it was too expensive.
- In the end, we used gradient descent to optimize a 5x5 kernel.
- However, it was not that much better than the Neville incremental predictor, which is what we are using today since it does not require any training.

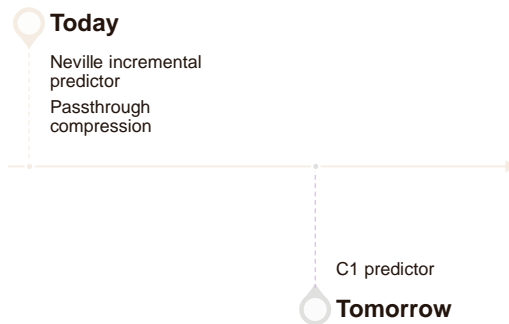
→ Lossless Compression



34:00

- The purpose of the predictors is to predict the height value at the various texels.
- But we need the reconstructed height to be lossless.
- So, we save the delta between the predicted value and the true value in an error stream.
- That error stream tends to be quite sparse, so it compresses well using simple compression techniques.
 - All results thus far have included this compression.

→ Virtual Height Map



34:30

- Using Neville incremental, it takes about 1ms to uncompress a 68x68 page on PS4. We will certainly re-evaluate this, and we are likely to switch to just the C1 predictor, since it performs close to Neville Incremental on our data and should be quite a bit cheaper.



Blend vertex height
towards terrain

Blend color
towards VT



35:00

- Using UHM or VHM, we can now implement skirts. Skirts allow meshes to seamlessly blend with the terrain.
- Skirt is 2 things at once:
 - 1st, the vertex height is altered to make them match the terrain height.
 - Artists control how far above & below morphing takes place.
 - 2nd, when shading meshes, we can sample from the virtual texture as a function of how close to the terrain the pixel is.
- In the bottom picture, we can see rocks seamlessly blend with the terrain.
- Clutter also makes use of the skirt tech. In the top picture, vertex morphing & color blending are disabled. You can see that the grass actually floats above the terrain.
- When vertex morphing is enabled, grass blades snaps to the terrain.

→ Clutter Tinting



Sample color
tint from VT



35:30

- Clutter, such as grass, can be color tinted so that it blends seamlessly with the terrain underneath.
- We use a lower resolution VT when color tinting, to get an average color.
- We can compute a color tint per clutter instance, per vertex or per pixel. This is at the discretion of content creators.

→ Mesh Tinting



Meshes color tinted by low-rez VT



- 36:00
- Related to clutter tinting is mesh tinting. By sampling very low-resolution VT following specific rules, we can make the same mesh look unique and better fit its environment.
- The cliffs on the right use very low resolution VT as a color tint to make them look unique.

→ Unified Terrain Surface (UTS)



Unified Terrain Surface

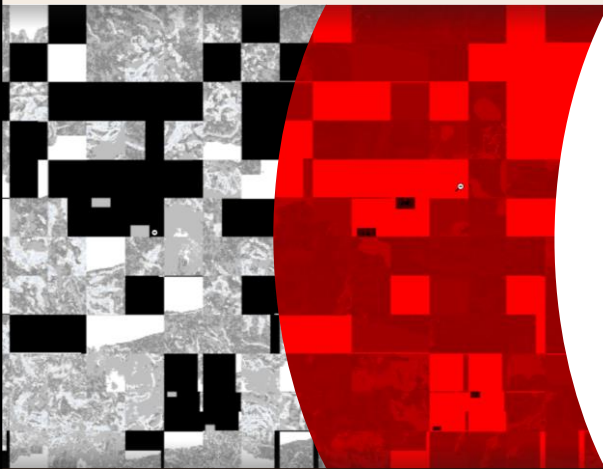
- 1 terrain surface instead of many
- Cracks a problem of the past
- Cleaner implementation



36:00

- We are currently working on new technology we call Unified Terrain Surface
- Instead of multiple terrain surfaces, we have 1 terrain surface that covers the entire world.
- It solves an annoying problem that we had before, when 2 terrain surfaces with different vertex density are adjacent to each other
 - That would create cracks in the ground and artists would often have to place meshes to hide them
 - Using UTS, this is not possible anymore
- With the Unified Terrain Surface, we also implemented a Unified Index & Color Map to go with it.

→ Virtual Index & Color Map



Reduce memory cost

Index & Color in sync

Not in sync with VHM

256x256 page (likely)

Gated compositing

Format

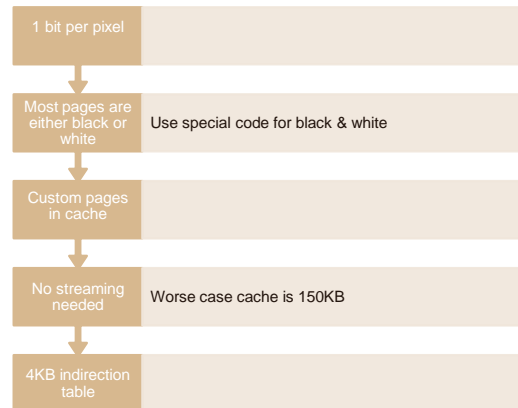
- Index: 16 bits
- Color: BC1



36:30

- After the process of unification, we can move onto the process of virtualization.
- Virtualizing the unified maps reduces their memory footprint, the major downside to unification.
- Because the index map cannot be interpolated, as each value is quite literally a terrain layer index, we use point sampling to generate the lower resolution mipmaps
- Since a pixel of the index map is meant to work with only the corresponding pixel of the color map, we also point sample the color map
- Finally, the resolution of the index & color map must match. This is again because a pixel of the index map is only meant to work with the corresponding pixel of the color map
- Virtual Index Map is currently 16 bits, but we have plans to reduce that to 10 bits. Virtual Color Map is BC1 compressed

→ Virtual Cutout Map



37:30

- Beside virtualizing the index & color map, we also virtualize the cutout map.
- A cutout map is quite simple as it is only 1 bit per pixel.
 - Either the sample is cut out or it isn't.
- The vast majority of pages are white, we have some black pages, and a few pages have a mix of cutout & non cutout pixels.
- The Virtual Cut Out Map uses an indirection table like the other virtual maps do.
- For the black & white pages, we simply use a special index to signify black or white. We only put in the cache, pages that have a mix of cutout & non cutout pixels.
- The data is so small, we don't even bother with down sampling.
- Since we can only have up to 254 such pages, it means the worst-case memory cost for the virtual cut out map is 150KB.
- On our large maps, the memory cost for the indirection table is a meager 4KB.

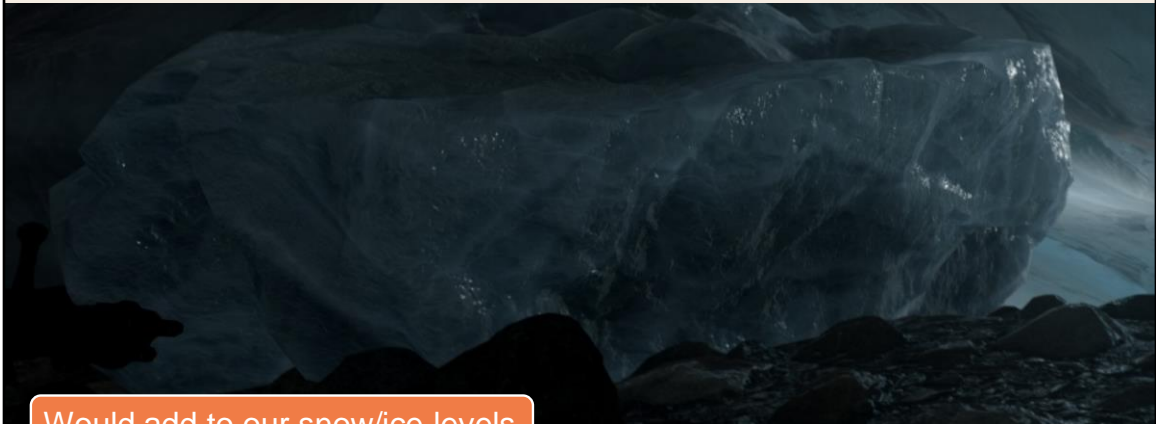
→ Vista UVs & VT Don't Mix



38:30

- Vista UVs are great at making a terrain layer look good both close & far to the camera.
- The issue is that it relies on camera distance. We alpha in macro details and alpha out micro details when the camera moves away from the surface.
- The reason we do that is because macro contribution looks good at a distance, but pixelated when seen up close.
- With virtual texturing, the use of the camera distance is problematic, since VT has no concept of camera.
- We have to fake it by translating a mip level to a camera distance. For example, mip 0 is meant to be seen from a max distance of about 5 feet. Mip 1 doubles that distance, while mip 2 doubles that yet again.
- It creates a discontinuity in the mip chain of the VT pages, since now, 2 adjacent mips will effectively have slightly different data.
- In practice, we have been able to make this work in a lot of places, but we are still exploring ways to remedy this problem.

→ Parallax Mapping



Would add to our snow/ice levels

© 2023 Activision Publishing, Inc

ADVANCES IN REAL-TIME RENDERING IN GAMES COURSE

39:30

- For ice, we often use a cheap parallax mapping technique, that does a decent job at emulating depth.
- Parallax mapping isn't supported by our terrain technology though, but we see no reason why not.

→ 4th BC1 Texture



Needed to
enable multiple
features

• Scattering + Glint + Parallax



39:30

- We are currently adding a 4th BC1 texture.
- This will give another 3 channels to play with for a grand total of 4 optional channels.
- Scattering and glint are often used on snow. Currently, we can pick one or the other, but you cannot have both.
- Similarly, parallax is often used on ice, hence on our snow levels. With a 4th BC1 texture, we could have scattering, glint & parallax all at the same time.

→ Procedural Decals



Cheap & quick way to add details



© 2023 Activision Publishing, Inc

ADVANCES IN REAL-TIME RENDERING IN GAMES COURSE

40:00

- We are experimenting with procedural decals, that is decals that have 0 memory cost. They can be useful in adding details to an otherwise boring terrain layer.

→ Dynamic Decals



For
things
on snow
or sand

- Footsteps
- Treads

Dirty VT
region

- Already
implemented
for editing



40:30

- Virtual texturing is a static cache, but we believe we could have some amount of dynamicity.
- A great example would be footsteps on snow or sand.
- All the tech is already there to automatically recompute a virtual texturing region.
- We just need to try it out and see how well it works.

→ Large Map Events



Needed
to
support
big
events

- E.g: Nuke section of the map

Mark
section
of the
map as
dirty

- VT updates itself
- Virtual maps update themselves
- Memory cost is minimal



40:30

- We recently added the ability to trigger large scale events to dramatically change some area of our map.
- When it comes to terrain, we want the geo to be affected.
- In this example, an airplane crashed on the ground.
- Using the virtualized maps, implementing this is easy and cheap, since only the pages that are affected need to be updated.



Thanks to all the Devs



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

Alexis Benamira
Ben Menke
Daniel Fetter
David Christie
David Perkins
Frank Luna
Jared Krahn
Jiyu Huang
Josh Lawrence
JT Hooker
Logan Bowers
Ryan Sammartino

And many others,
at High Moon and across all of Activision!



© 2023 Activision Publishing, Inc

ADVANCES IN REAL-TIME RENDERING IN GAMES COURSE

41:00

Many fine folks participated, one way or another, on the work presented here. From the bottom of my heart, I want to thank them all.