
GDC
Object Space Lighting
Dan Baker
Founder, Oxide Games
GAME DEVELOPERS CONFERENCE March 14–18, 2016 · Expo: March 16–18, 2016 #GDC16
UBM

Ashes of the Singularity

OXIDE

Stardock

Nitrous Engine/Oxide Games

- New studio founded from industry vets – Firaxis, Zenimax, Stardock
- Ground up, custom engine
- Several titles in development using Nitrous, including Star Control

Types of Renderers

- Forward Renderer
  - Many DX8-DX9 titles
  - Still many DX11 titles
  - Unreal 3.0
- Deferred Render
  - X360+, PS3, DX11
  - Unity, Frostbite
- Forward+ Renderer

Forward Rendering

- Forward rendering, triangles are rasterized as they would appear directly into the back buffer
- Lighting, and shading occur at time of rasterization.
- “Just in time shading.”

ForwardPros/Cons

- Pros:
  - Straightfoward
  - MSAA works works well
  - Decouple systems, easy to add in new material types
  - Alpha blending is simple
- Cons-
  - No shader anti-aliasing
  - Lights are (typically) bound with the materials

Deferred

Defered Renders, shading does not occur during rasterization

Instead, important shading parameters such as normal, albedo, position, etc are saved into a deep back buffer

The back buffer is then shaded

Deferred

- Pros
  - Stable performance characteristics
  - Lighting can be semi-decoupled from shading
  - Lots of techniques like SSAO can be used
- Cons
  - Terrible intrinsic aliasing problems. Not solvable
  - Limited number of materials possible
  - Alpha blending is complex or impossible for certain classes of materials

Forward+/Hybrid

Forward + is a general term for hybrid techniques. Forward+ renders use multiple passes to calculate some of the scene data that is used in Deferred

Main rendering is done as forward+

Most engines are hybrid

Forward +

- Pros
  - Most of the advantages of a forward renderer
  - Can also decouple lights from materials
- Cons
  - Still doesn’t solve aliasing problems related to shader aliasing
  - Complex to implement

Object space

- DX12 Renderer!
- In deferred, shading takes place after rasterization
- In forward and Forward +, shading occurs during rasterization
- In object space, shading occurs **before** rasterization

Problems trying to solve

- Shader anti-aliasing/Stability
  - High spec powers and discontinuities in shaders plague us.
  - Longer shaders get, the more this is a problem
  - LEAN mapping and other techniques are not extensible to more complex materials
- Shade frequency may need to be higher then screen space to get good results
- Shading can be done at different temporal frequency then rasterization
- Materials can be layered
- No limit to number of materials used
- Stable performance – maximum number of shaded samples
- Shading alpha blended materials is important

REYES

Render

Everything

Your

Eye can

See

Object Space Lighting

- Inspired by REYES
- High level concept – shading occurs first
- The SPACE of shading is completely different
  - Intrinsically better mathematically
- Scene is constructed from shaded sample points
- Details from REYES are very different

Before we begin

- AotS is a physically based renderer
- Underlying Nitrous engine has many services for PBR, but doesn't intrinsically force you to be
  - Completely determined by the PBR nature of materials of application. All power rests on the materials
- Post process services

The Basics

- Game allocates several large texture sheets
  - 4kx4k, 16 BPP
- Object rendering broken into 2 passes
  - Shading pass
  - Raster pass
- Lighting system works similar to Forward + Style

Random things we can do

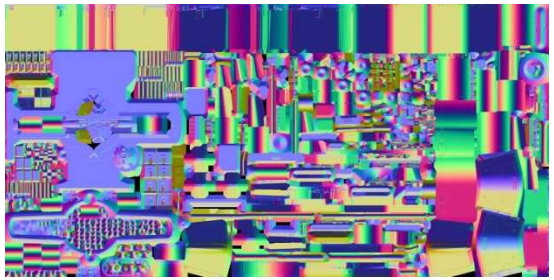
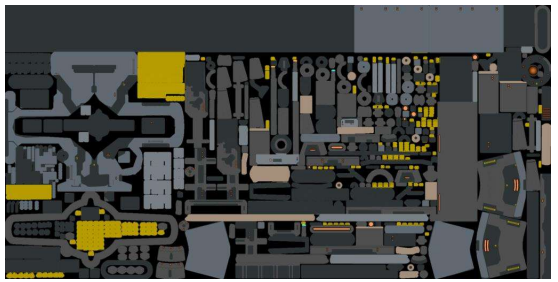
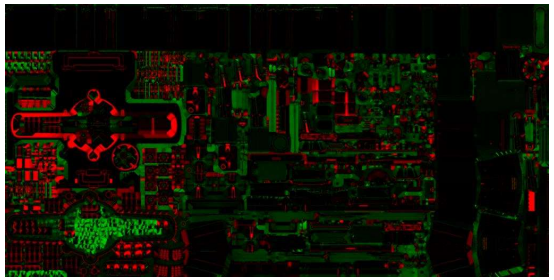
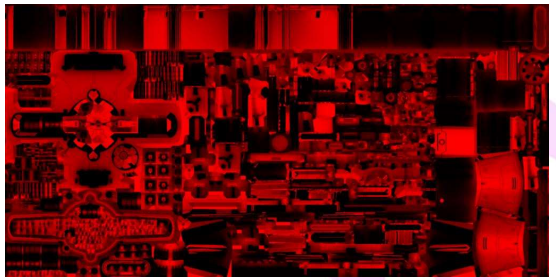
- Attach preconvolved BRDF lighting environments, called a Axial light
  - All metals are true specular, no diffuse at all (can't be done on deferred because need a unique 3d texture for each material)
- High spec powers, not a problem
  - 1000, 2000x spec powers are stable and look good

Challenges

- All objects must be chartable
- Objects which are close on screen have hugely uneven texel density
- Quality of Normals matters with high spec powers – Authoring more precise normals takes effort
  - Not an issue if you stick with lower spec powers
- No Vertex Data during shading

What an asset looks like

- Normal Map (world space)
- Position Map (lower res)
- AO Map
- Material Mask
- Textcoord Remap

2 Passes

- All objects have a material group
- Material Group consists of
  - Materials, as many as artist wants
  - Rasterization shader
- Important: Animation must be done twice
  - Once during shading to get a position/normal for shading
  - Once during rasterization

Material System

- Internal system called Oxide Shading Language (OXSL)
- Build on top of HLSL compiler
- Compiled into C++ components during application compilation

Material System

```
MaterialGroup PHCUnitMaterials
{
     DirectRenderResources = ::OSLEngineShaders::OSLResolvedTextures;
     DirectRenderVS   = DirectRenderVS;
     DirectRenderPS   = FinalAlphaPS;

     MaterialAppResources = PHCUnitsShadeTextures;
     MaterialAppParams    = AshesShadeData;
     MaterialAppGlobals   = ::OSLEngineShaders::OSLGlobalParam
     VertexData = AshesVertexData;

     MaterialGroupResources  = PHCUnitResources;
     MaterialGroupResources2=  ::AshesAxialLights;


     MaterialOSLResources = ::OSLEngineShaders::OSLResources;

     CodeBlocks =  OSLStandardDirectRenderShaders, DestructionMasks, ConstructionPS;
}
```

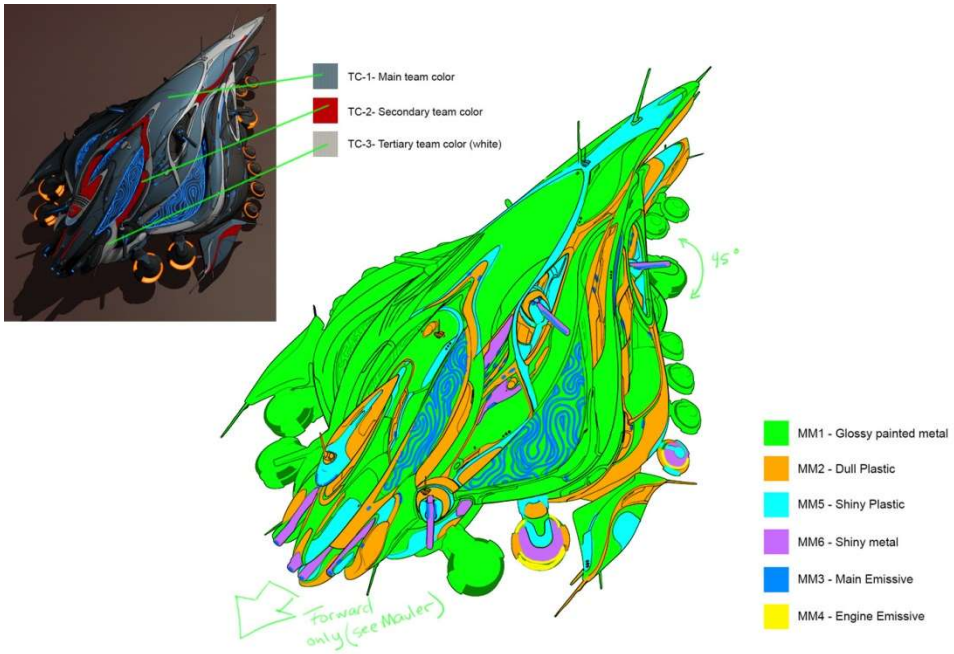
These shaders Rasterize to the screen using the shaded Object

Materials

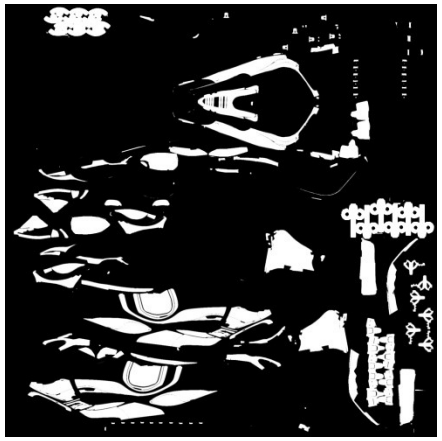
```
Material PaintedMetal : PHCUnitMaterials
{
CodeBlocks    = ::OSLEngineShaders::OSLBase,::OSLEngineShaders::LightingFunctions, ::AshesCommonFunctions::AtmosphereFunctions;
MaterialShader    = ShadePaintedMetal;
}

Material BareMetal : PHCUnitMaterials
{
CodeBlocks    = ::OSLEngineShaders::OSLBase,::OSLEngineShaders::LightingFunctions, ::AshesCommonFunctions::AtmosphereFunctions;
MaterialShader    = ShadeMetal;
}

Material Emissive : PHCUnitMaterials
{
CodeBlocks    = ::OSLEngineShaders::OSLBase,::OSLEngineShaders::LightingFunctions, ::AshesCommonFunctions::AtmosphereFunctions;
MaterialShader    = ShadeEmissive;
}

Material EngineEmissive : PHCUnitMaterials
{
CodeBlocks    = ::OSLEngineShaders::OSLBase,::OSLEngineShaders::LightingFunctions, ::AshesCommonFunctions::AtmosphereFunctions;
MaterialShader    = ShadeEngineEmissive;
}

Material AnimatedEmissive : PHCUnitMaterials
{
CodeBlocks    = ::OSLEngineShaders::OSLBase,::OSLEngineShaders::LightingFunctions, ::AshesCommonFunctions::AtmosphereFunctions;
MaterialShader    = ShadeAnimatedEmissive;
}
```

Retributor

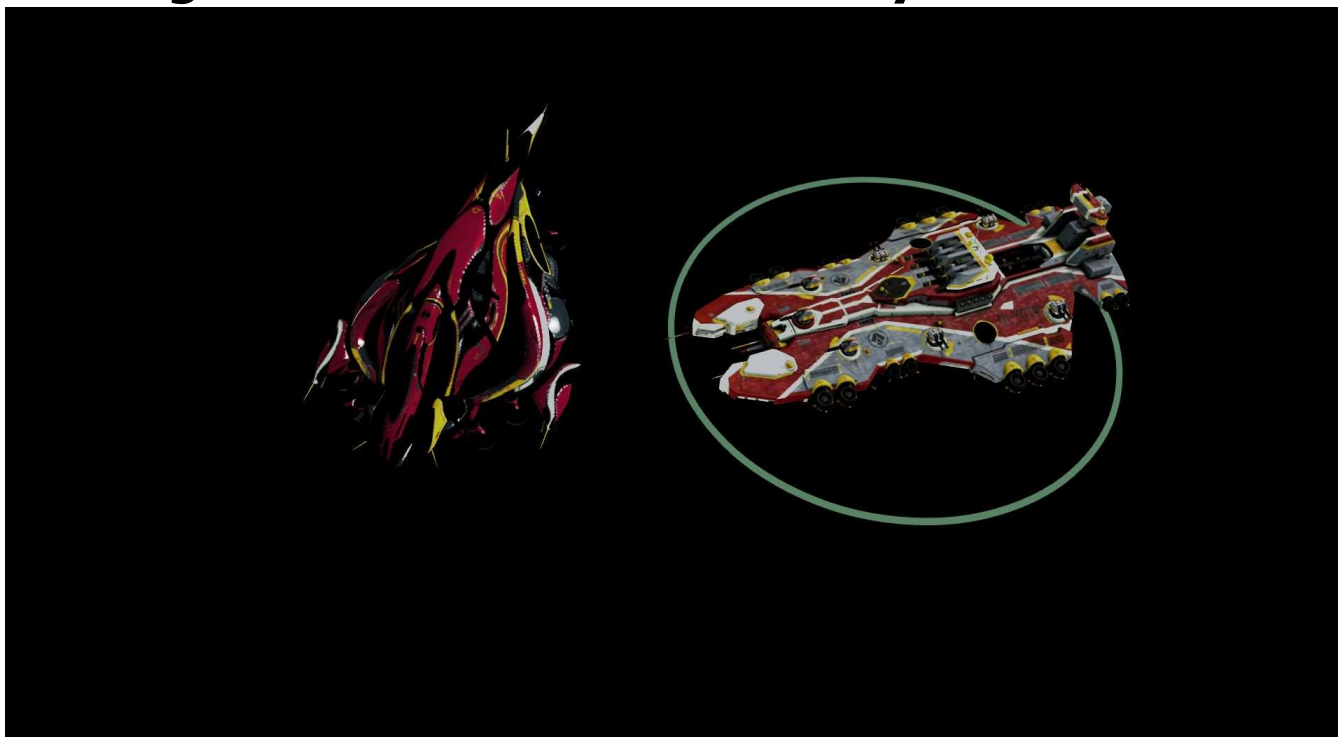
Objects Material layers



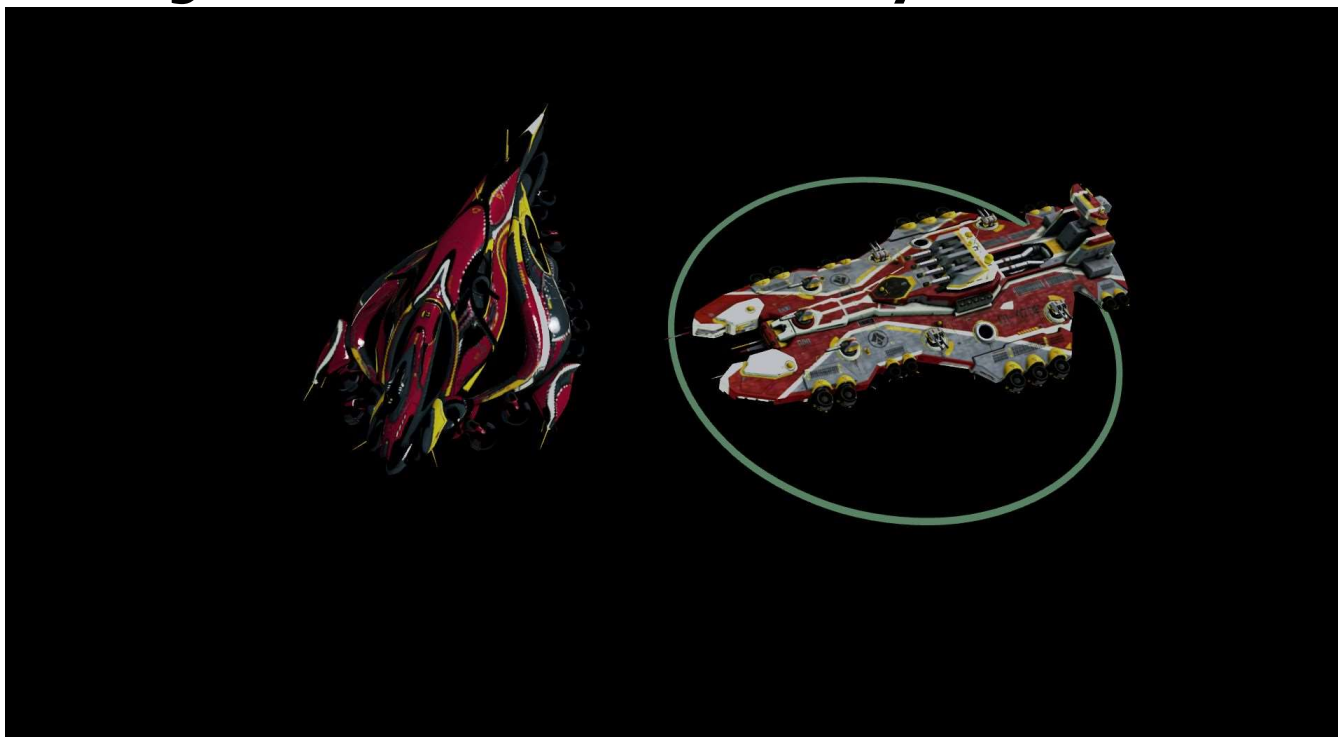
Material Masks

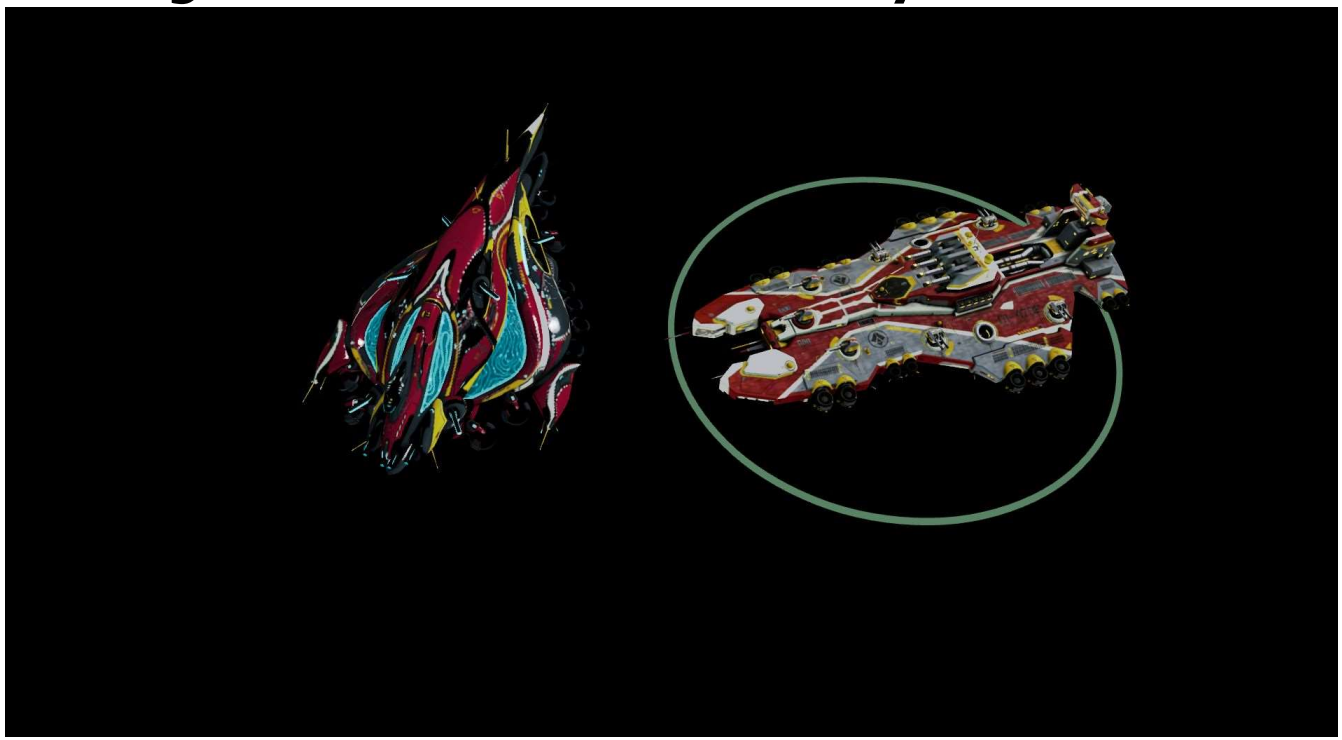

An object with it’s layers



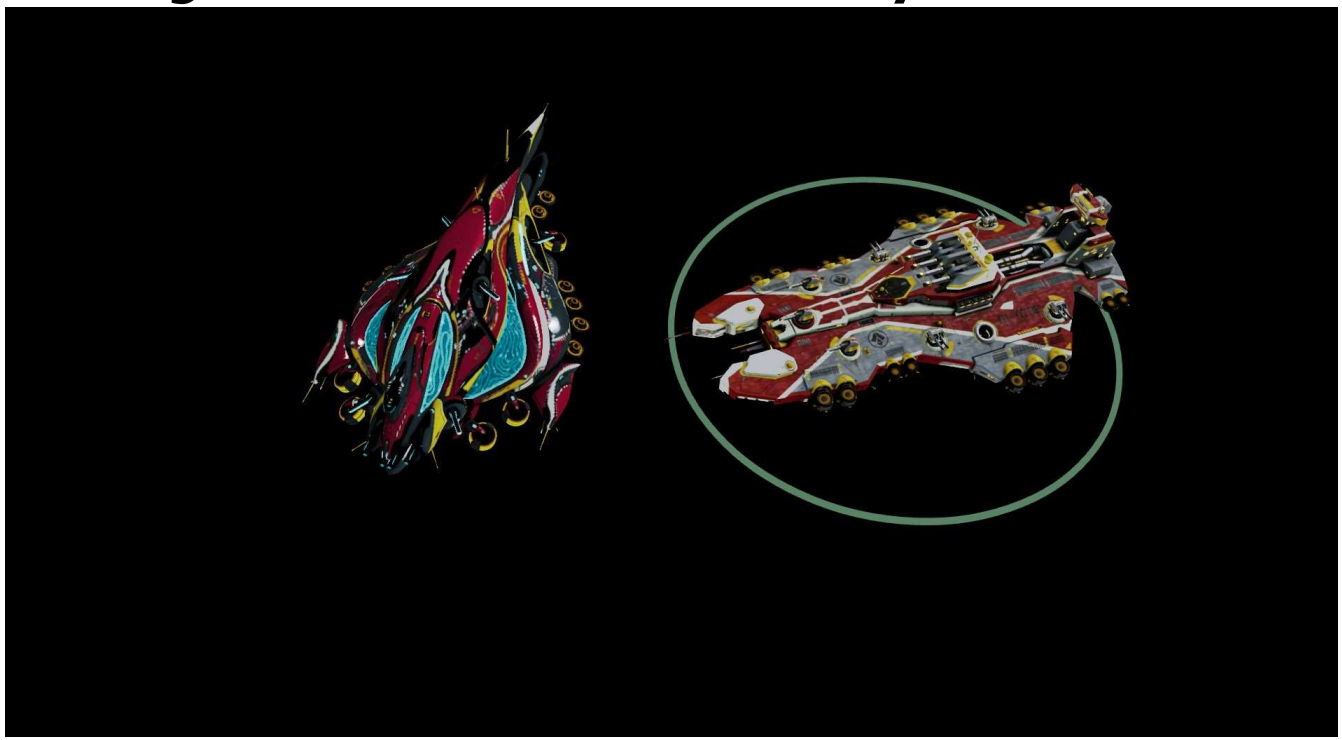
An object with it’s layers



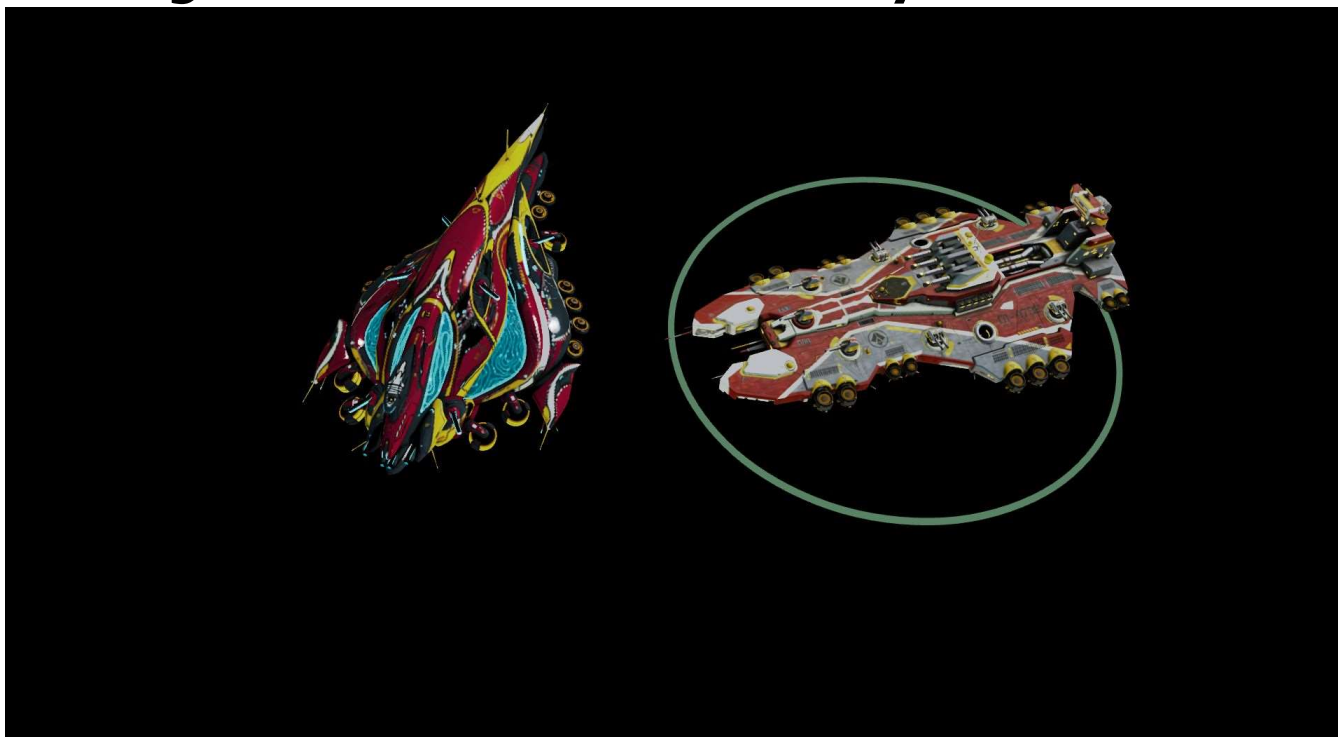
An object with it’s layers



An object with it’s layers



An object with it’s layers



Handling lights

- Forward + style
- Scene has small number of directional lights
- And can have large number of point/spot lights
- Materials responsible for iterating over lights

The light Matrix

- Lights in scene are placed into the light matrix
- Fixed level oct tree
- A light can appear multiple times inside the light matrix
- Very fast to use in shader, a handful of texture fetches to find the lights
- Shader iterates over the lights
  - Observation – iterating over lights in shader is no longer prohitive. Performance is good
  - Observation, performance is mostly related to bandwidth of loading the lighting information into shader, not making decisions about applying it
- Light Matrix is recomputed every frame
- Can handle many thousands of lights per frame with predictable performance impact
- Light matrix rebuilt every frame, about .3 to .5 ms to build

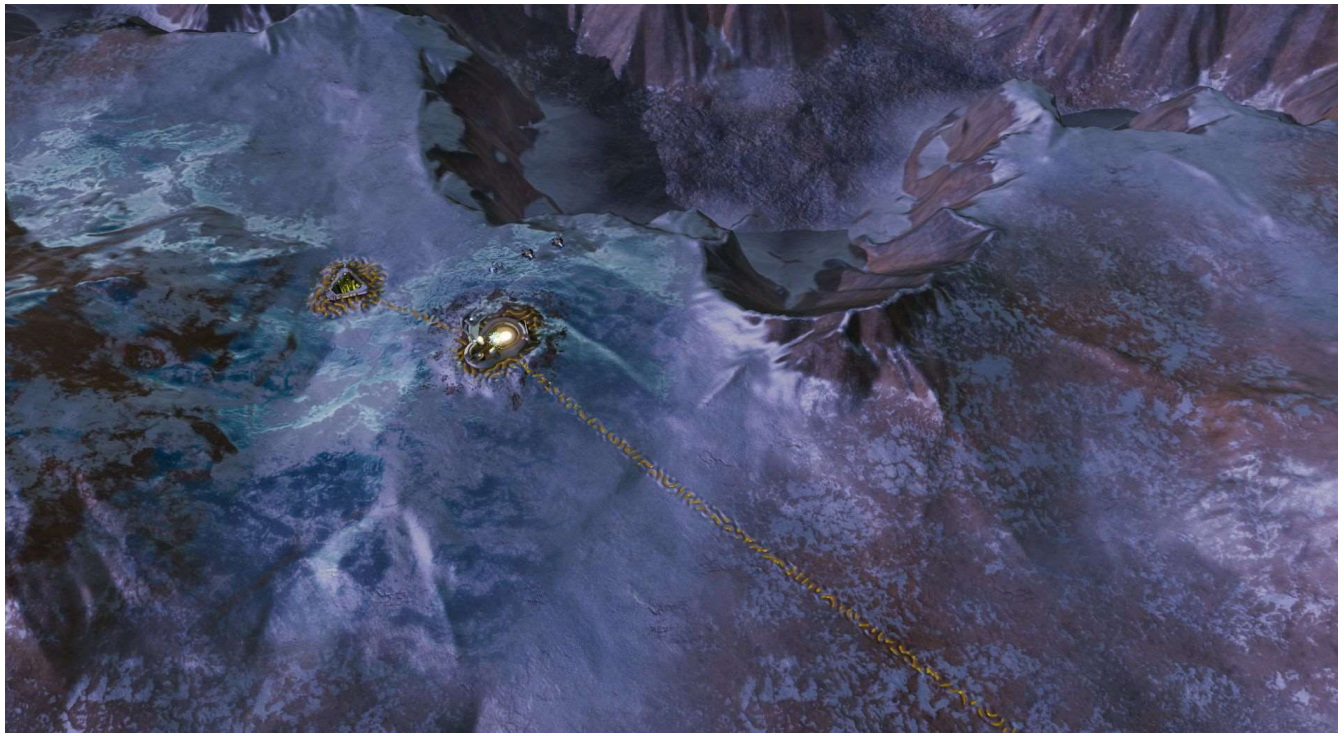
Overall process

- All objects in game are submitted for shading and rasterization. Queued for process
- During submission step, the estimated projected area of the object is calculated. Thus an object requests a certain amount of shading
- During shading, system allocates texture space for all objects which require shading. If the total request is more then available shading space, all objects are progressively scaled at shading rate until it fits
- Material shading occurs, processing each material layer for each object. Results are accumulated into the master shading texture(s)
- MIPS are calculated on master shading texture as appropriate
- Rasterization step: each object references the shading part step. No specific need that there is a 1:1 correspondence, but this feature is rarely used.

Issues

- Many draw calls can be issued for each object – each material layer is a batch. Then each object gets rasterized. Approx 2x number of batches as typical game engine
  - D3D12 very useful on cutting out overhead
  - Batches are relatively simple
- Allocation of object space can matter a great deal to image quality
- Decisions for total texel density are decided at an object level. Objects which have varying density will have issues.
  - Objects which are very large on screen
  - Terrain

Terrain Materials

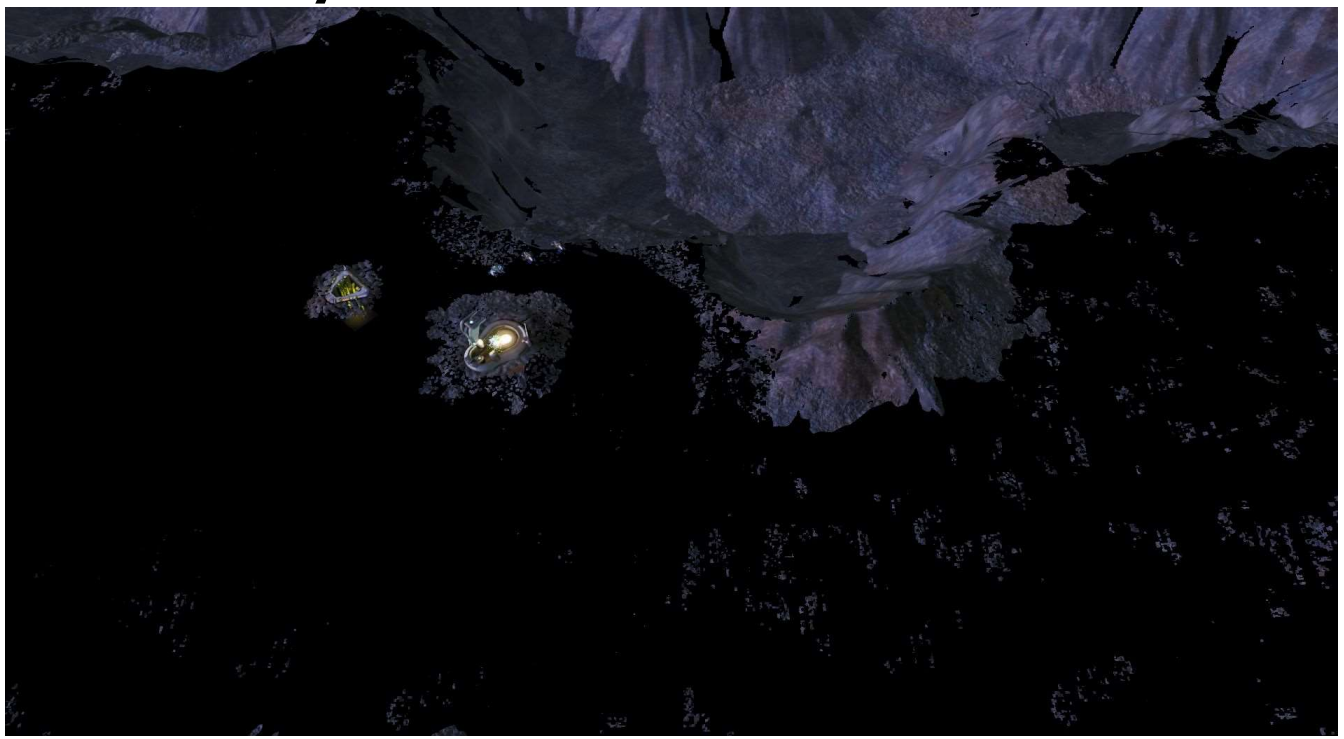


Terrain has some unique features

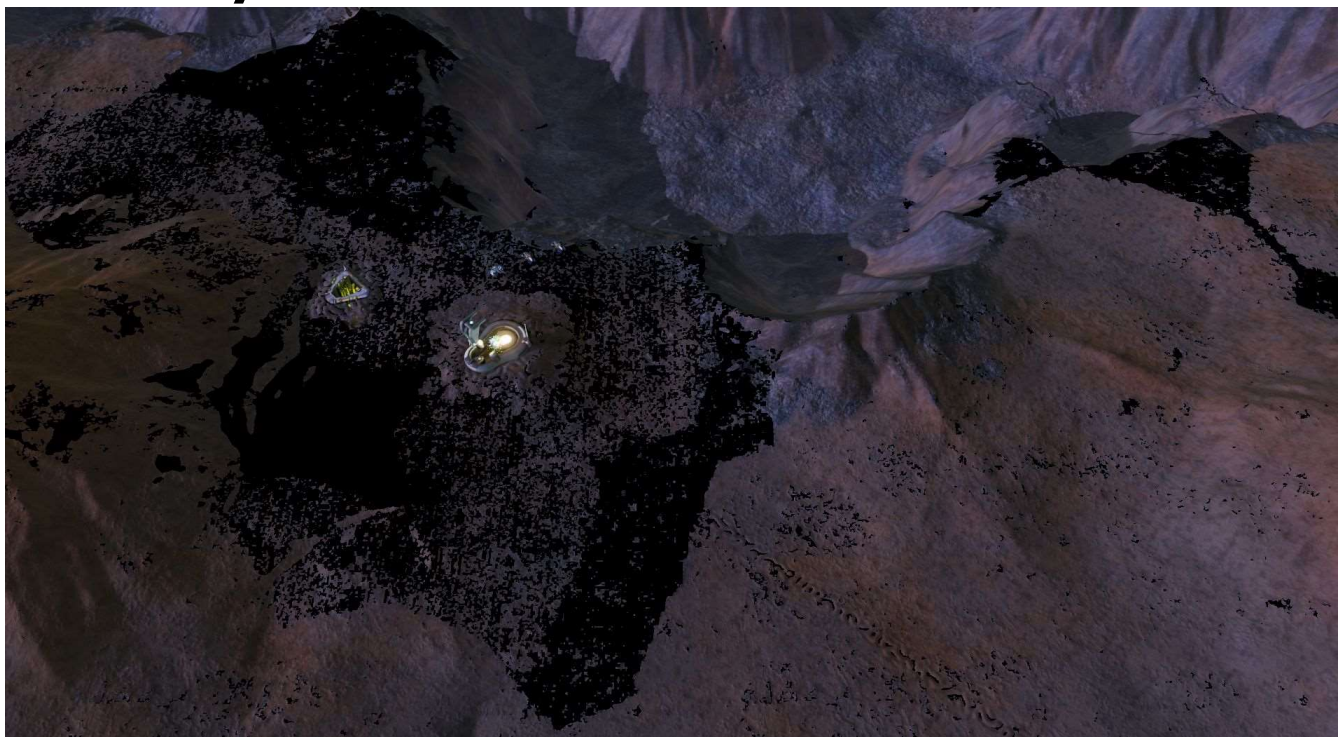
- Shaded before rasterization
- Allows the concept of a Pre-cal
  - Unlike a decal, a precal is generated before shading
  - Meaning each material layer can decide how to react to precal
  - Scorch marks primary use for this
  - Also used to create power lines on map

No Layers

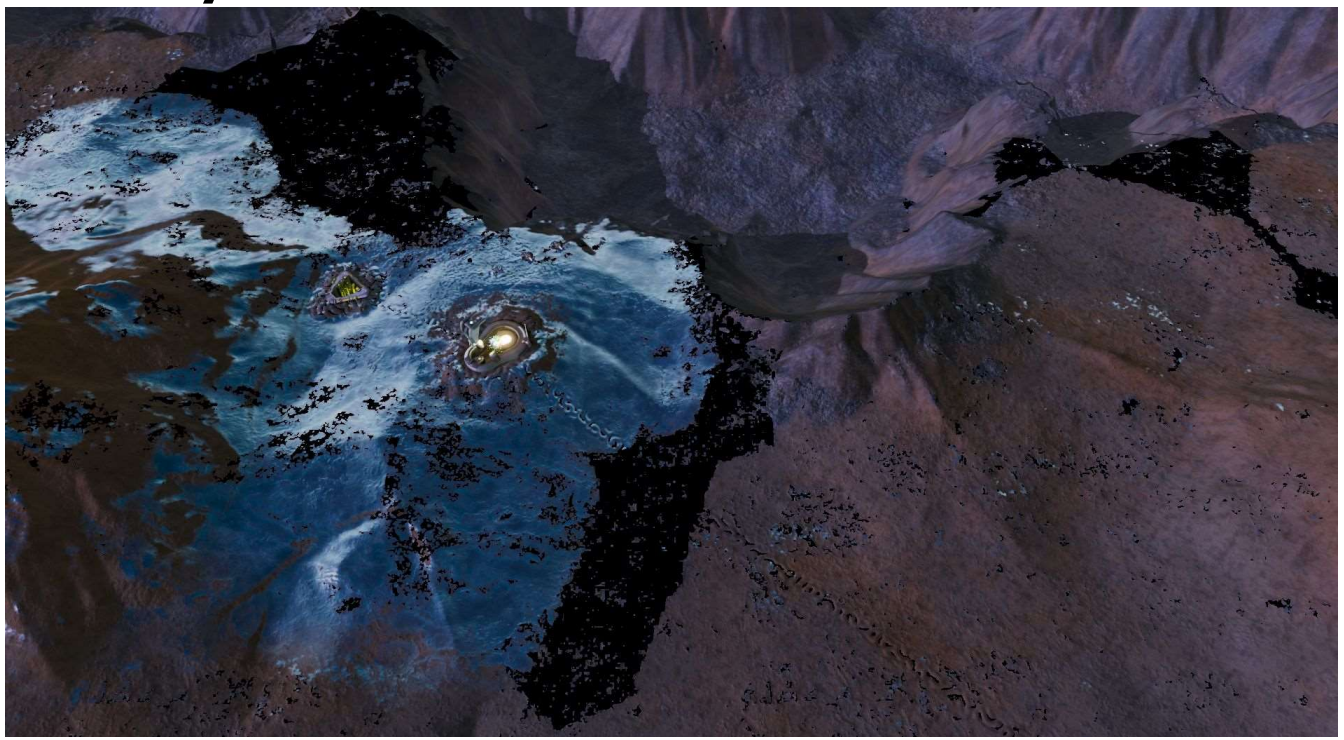
Basin Layer



Dirt Layer

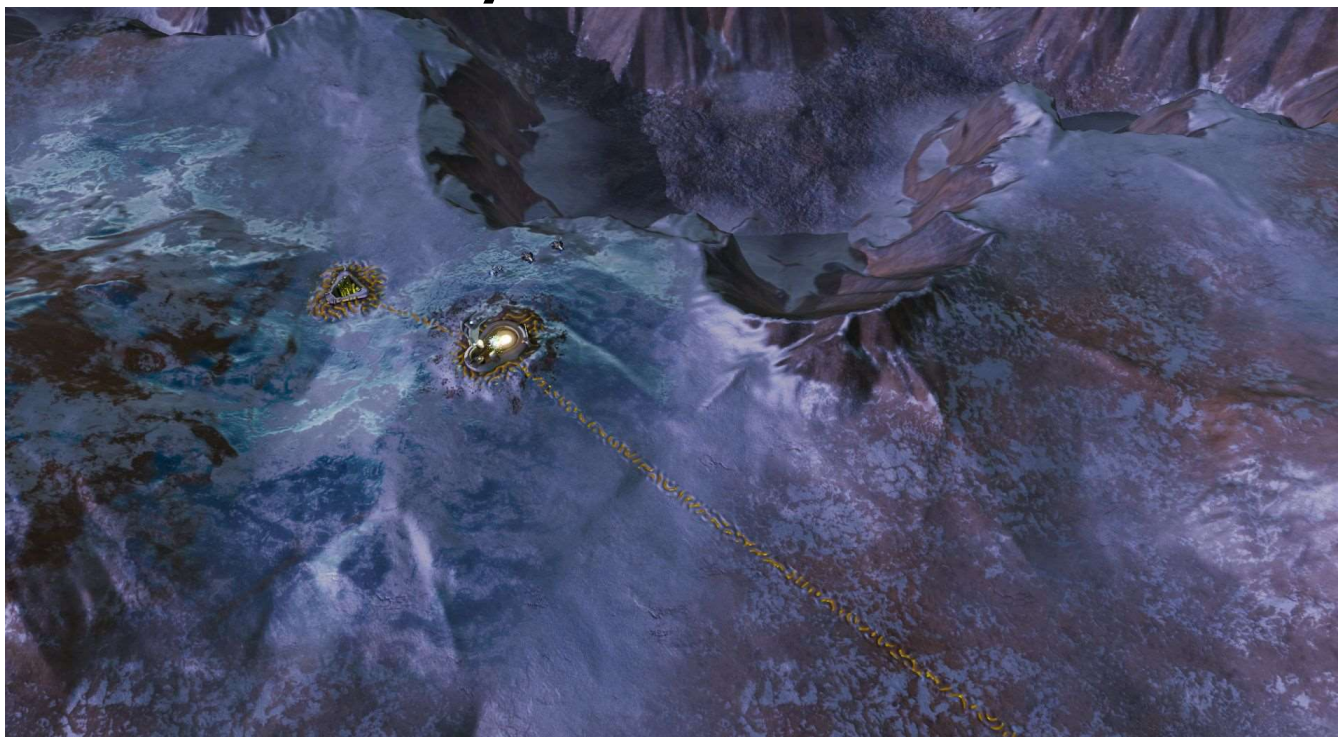


Ice Layer



Snow Layer

Turinium Layer



Terrain

- Terrain is a large continuous mesh
  - Can’t be shaded like an object
  - Could never afford to place the entire terrain into a texture – to much offscreen unused space
  - Texel density needs to vary across what’s on the screen
- Solution: Stitch Map

Stitch Map

- Terrain is going to be quilted together
- The terrain is made of little square patches
- Indirection in raster shader stitches the map back together
- Loosely similar to Perfect Hashing (Hughes Hoppe)
  - Except recalc every frame, and mesh is simpler

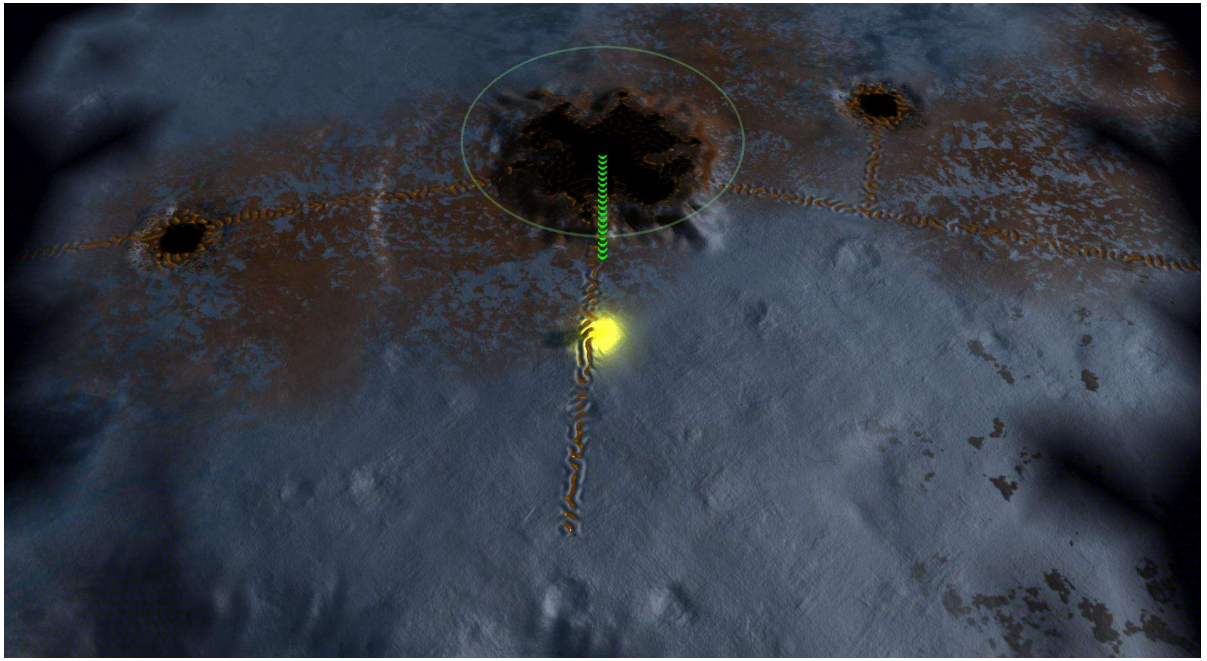
Generating the stich map

- Terrain has one giant texture where shading samples go into
- During a frame, each patch of the terrain is projected into an estimate for area
- Allocated into the shading texture through similar technique as Object Space renderer
- However, the tiles represent one continues mesh – Breaking it apart would cause obvious seems
- Stich map is generated

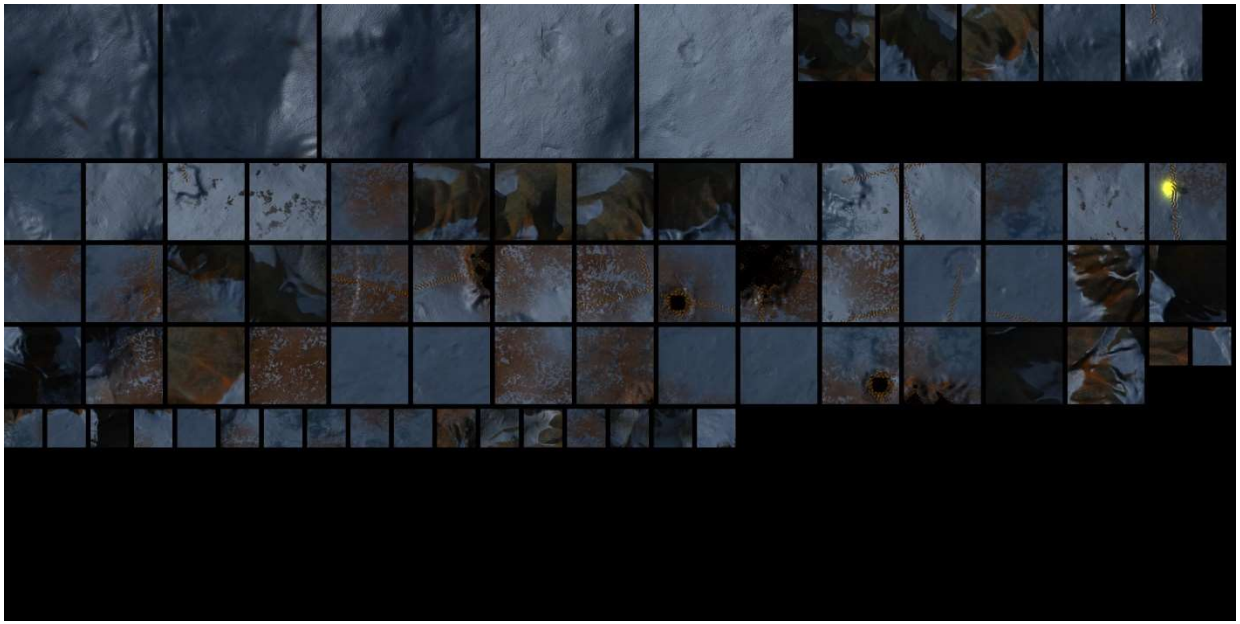
Variable density

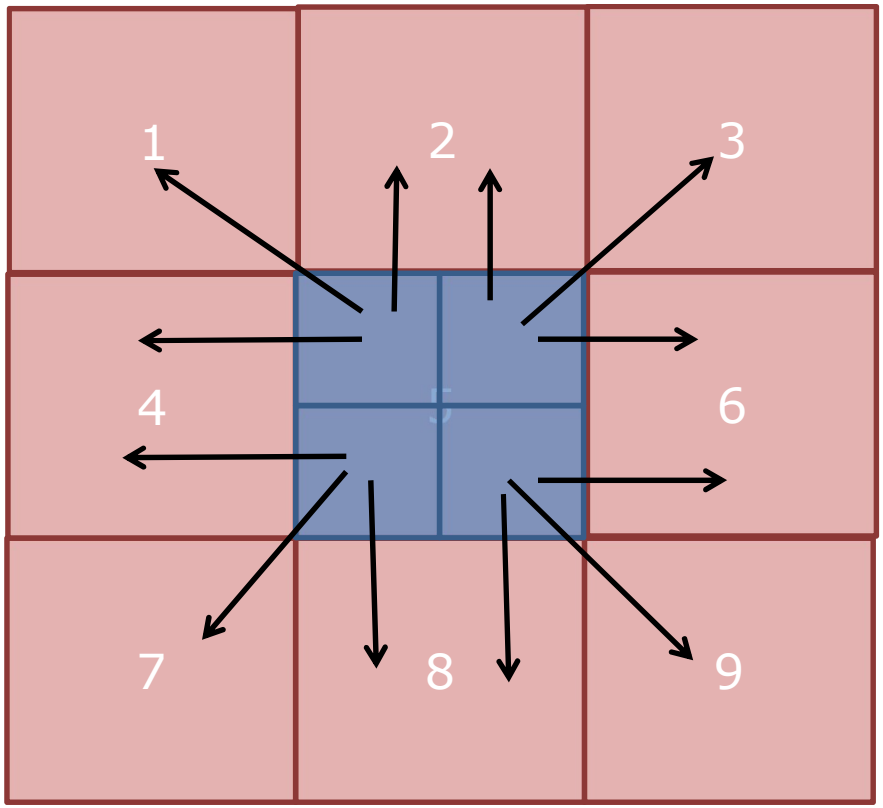
- Big trick is to allow each tile to have a unique resolution
- On the same screen, some tiles may be only 128x128 texels, while some could be 1024x1024
- Terrain system will allocate more texel space to regions of terrain which are vertical, and need more space to avoid stretching artifacts

Terrain Stitching



StichMap source





Each red cell points to an entirely different region of our shaded texture, resolutions may not even match

How do we blend between them on the edges?

Reconstructing the stitch map

```
ActualTexCoord[0] = MapTexCoord(PageMapLoc[0].xy, PageTexCoord + float2(PageMapOffset[0]),PageMapMIP[0]);
        ActualTexCoord[1] = MapTexCoord(PageMapLoc[1].xy, PageTexCoord + float2(PageMapOffset[1]),PageMapMIP[1]);
        ActualTexCoord[2] = MapTexCoord(PageMapLoc[2].xy, PageTexCoord + float2(PageMapOffset[2]),PageMapMIP[2]);
        ActualTexCoord[3] = MapTexCoord(PageMapLoc[3].xy, PageTexCoord + float2(PageMapOffset[3]),PageMapMIP[3]);
        ActualTexCoord[4] = MapTexCoord(PageMapLoc[4].xy, PageTexCoord, PageMapMIP[4]);


        float2 DerX = ddx(InputTexCoords * s_RemapTableDims * s_PageRemapTexCoordScale[0].xy);
        float2 DerY = ddy(InputTexCoords * s_RemapTableDims * s_PageRemapTexCoordScale[0].xy);

        float4 Sample1 = PageMapBackstore.SampleGrad(SS_DEFAULT, float2(ActualTexCoord[0]), DerX*fDirScale[0],
DerY*fDirScale[0] );
        float4 Sample2 = PageMapBackstore.SampleGrad(SS_DEFAULT, float2(ActualTexCoord[1]), DerX*fDirScale[1],
DerY*fDirScale[1] );
        float4 Sample3 = PageMapBackstore.SampleGrad(SS_DEFAULT, float2(ActualTexCoord[2]), DerX*fDirScale[2],
DerY*fDirScale[2] );
        float4 Sample4 = PageMapBackstore.SampleGrad(SS_DEFAULT, float2(ActualTexCoord[3]), DerX*fDirScale[3],
DerY*fDirScale[3] );
```

Material layers

- Object can have many layers, what if layer is sparce?
  - Discarding on a mask early is very fast, hardware good at aboring entire regions of a texture
  - Possible to get better perf by finding the regions for each material layer that are masked out, but in practice masking functions are complex and happen in shaders

Occlusion

- What about occluded pixels?
- In both forward and deferred renderers, one does not pay the cost of shading non visible samples
- In our engine, wastage for shaded objects
- Objects which are not visible are culled early in pipe

More efficient hardware

- Occlusion inefficiences in current design are masked by increased hardware efficiency
- No wastage on shading for small triangles, e.g. 2x2 stamp problem
- Overall shading throughput is far higher then a forward renderer.
- Don’t have to pay maximum cost of an uber shader like deferred
  - Simple materials are fast

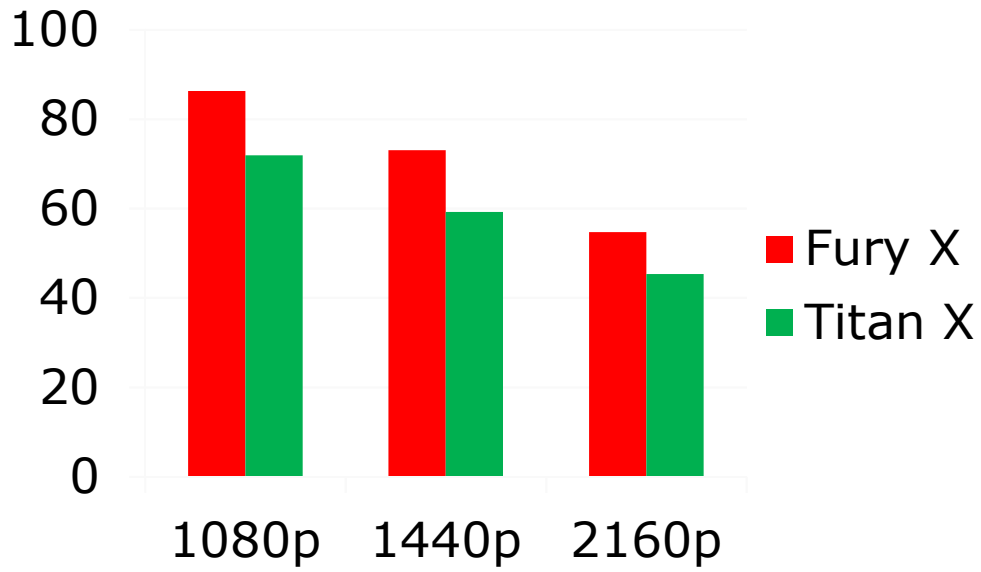
Other notes

- Any object can bypass the object space lighting system
- Trees, particles which have very simple shaders get no benefit out of it, thus default back to forward
- Can be intermixed easily with forward renderers

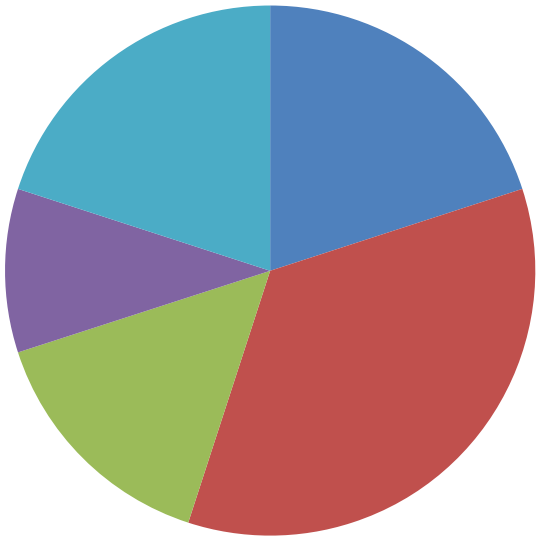
Results

- Initial design point – little doubt in overall quality improvements
- Primary concern was performance
- Great knobs for controlling performance
  - Shading Samples
  - Terrain Shading Samples
- Not as dependent on resolution for scaled performance

Performance at Resolution



Where our frame time goes



- Unit Shade
- Terrain Shade
- Rasterize
- Lighting/Shadow Setup
- Other/Post

Some obvious observations

- Shading does not need to occur at same temporal frequency as rasterization
- 30 fps for shading is adequate
- 60 fps for smooth camera, 90 fps for VR

The future: Async Shading

- With DX12, apparent that shading can be completely decoupled from rasterization
- Using async compute
  - Slow downs of shading would not impact FPS
  - Will use GPU more efficiently
- Early tests are promising
  - But will need to be DX12 exclusive

Challenges

- Dealing with large meshes which have varying density on screen a challenge
  - Need to apply stich mapping on more then just terrain
- Occlusion system would be needed for an FPS style game
  - Not needed for strategy games
- Art pipeline requires that all objects be charted with unique UV map
  - But already had this requirement for art tools
- Integrating any screen space techniques

Obvious future directions

- With shading of scene captured,
  - Second bounce lighting?
- Simplified rasterization step, cost of shading is a non factor
  - Point rendering?
  - Stochastic reconstruction?