SIGGRAPH2009 NEW ORLEANS

Rendering Technology at Black Rock Studio

Jeremy Moore / David Jefferies

Advances in Real-Time Rendering in 3D Graphics and Games

Talk Contents

Part 1: Foliage Rendering in Pure

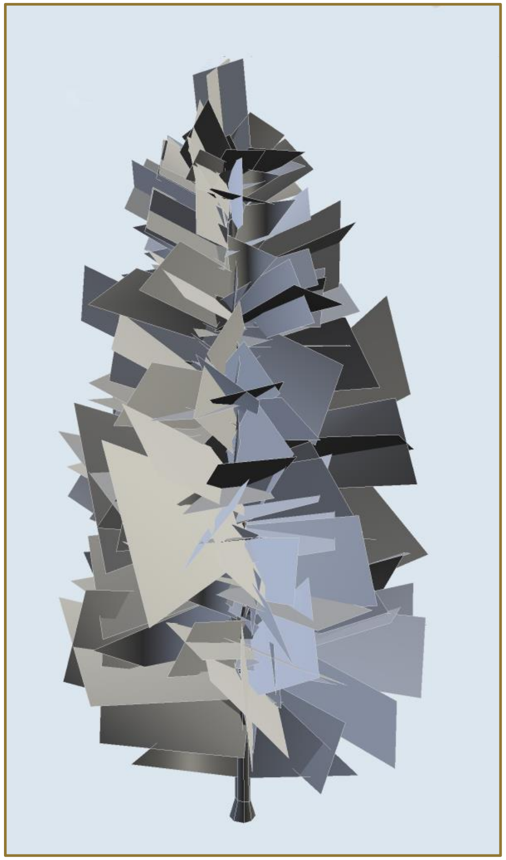
Talk Contents

- Requirements for Pure
- Alpha Compositing Basics
- Ground Cover Rendering in Pure
- Tree Rendering in Pure
- Screen-Space Alpha Mask Technique

Pure Game Footage

Pure Game Footage Minus Foliage

Foliage Rendering Using Alpha Textures



Alpha Compositing Basics

- Alpha blending
- Alpha testing
- Alpha to coverage

Alpha Blending

- Blend rendered fragment with destination pixel data according to a scalar blend value

```
result = (1-alpha)*destination + alpha*source
```

- Requires a read/blend/write operation
- Operation is not associative
  - Rendered objects should be sorted
  - Especially when combined with z-buffer use

Alpha Blend Depth Ordering

Alpha Testing

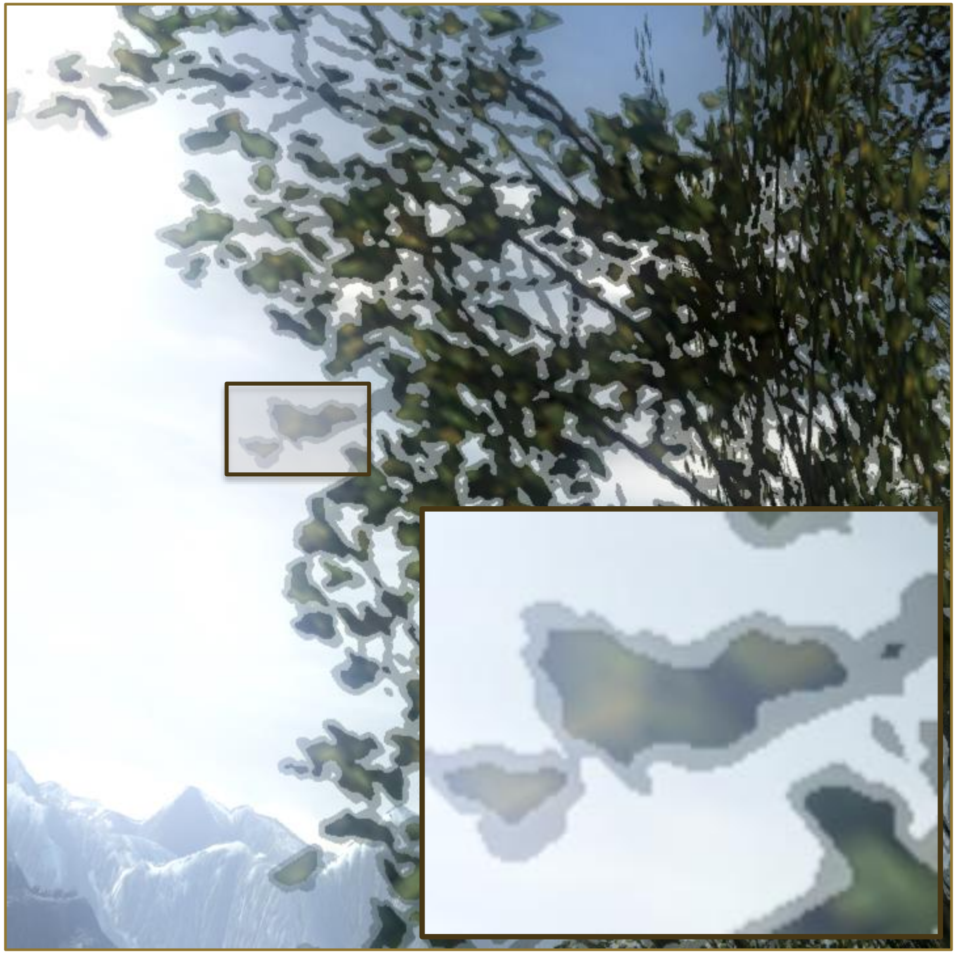
- One bit value determines if fragment is visible
- Works with z-buffer
- Can cause aliasing

Alpha Test Aliasing

Alpha To Coverage

- Converts alpha into a coverage mask for the pixel
- Coverage mask is AND'd with MSAA coverage mask
- Gives softer edges when combined with alpha testing
- Works with z-buffer
- But… the resulting alpha gradients don't always look great

Alpha To Coverage Aliasing



Ground Cover in Pure

Ground Cover Overview

Offline

- Density map
- 2D map used to place cover

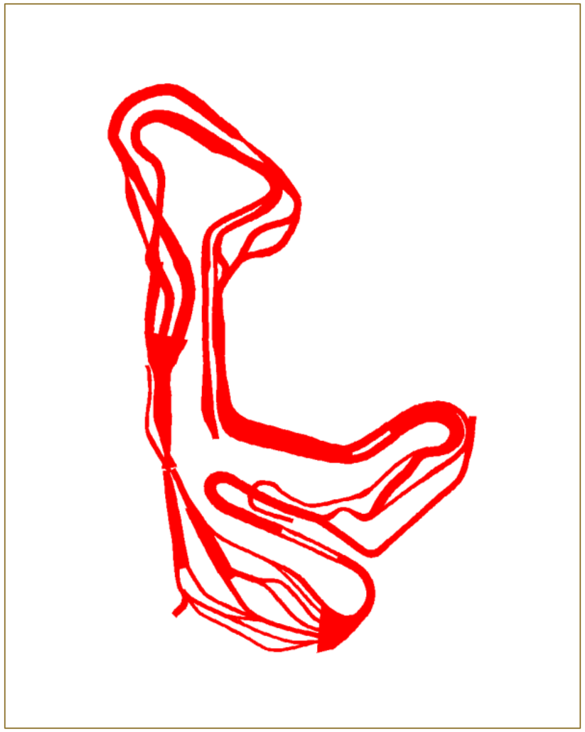
Camera movement

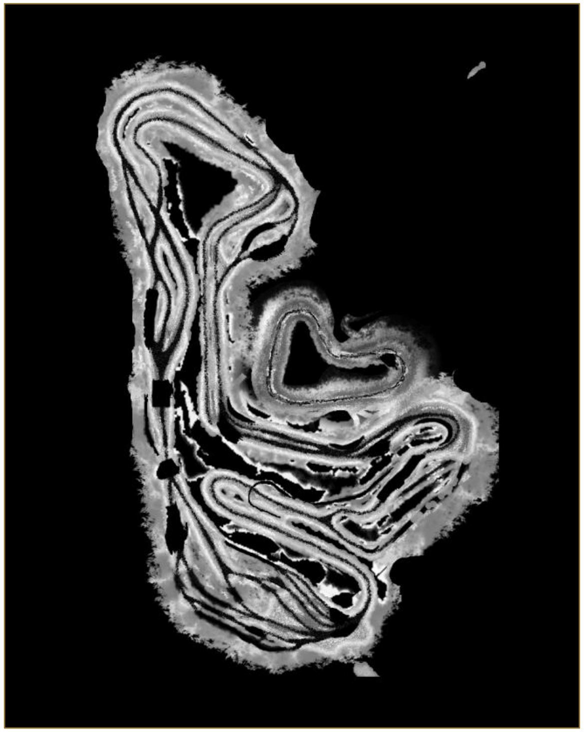
- Tile cache
- Calculated cover type/position/size

Per frame

- Vertex buffer
- Sprite vertices encoding type/position/size

Ground Cover Placement

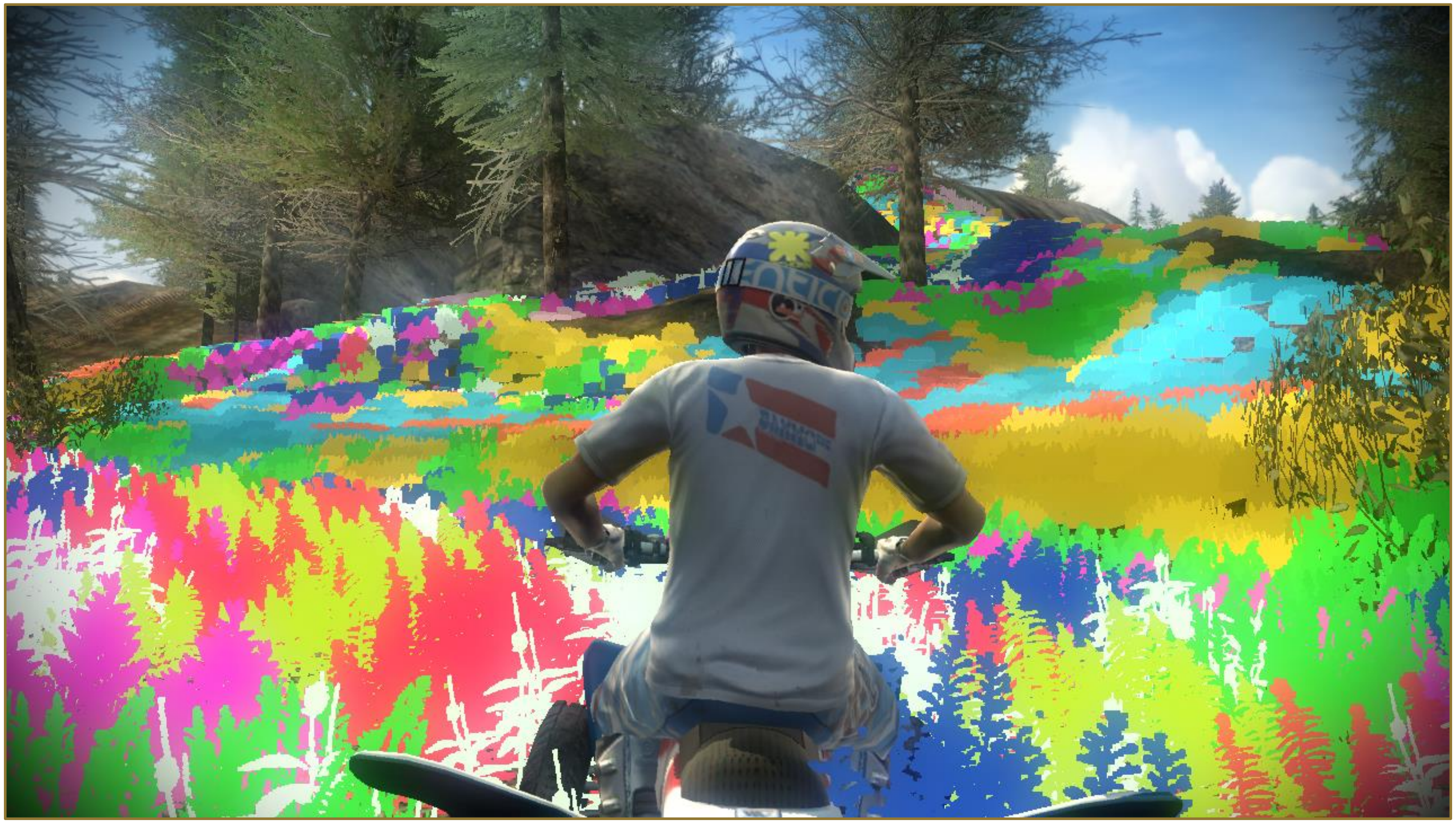




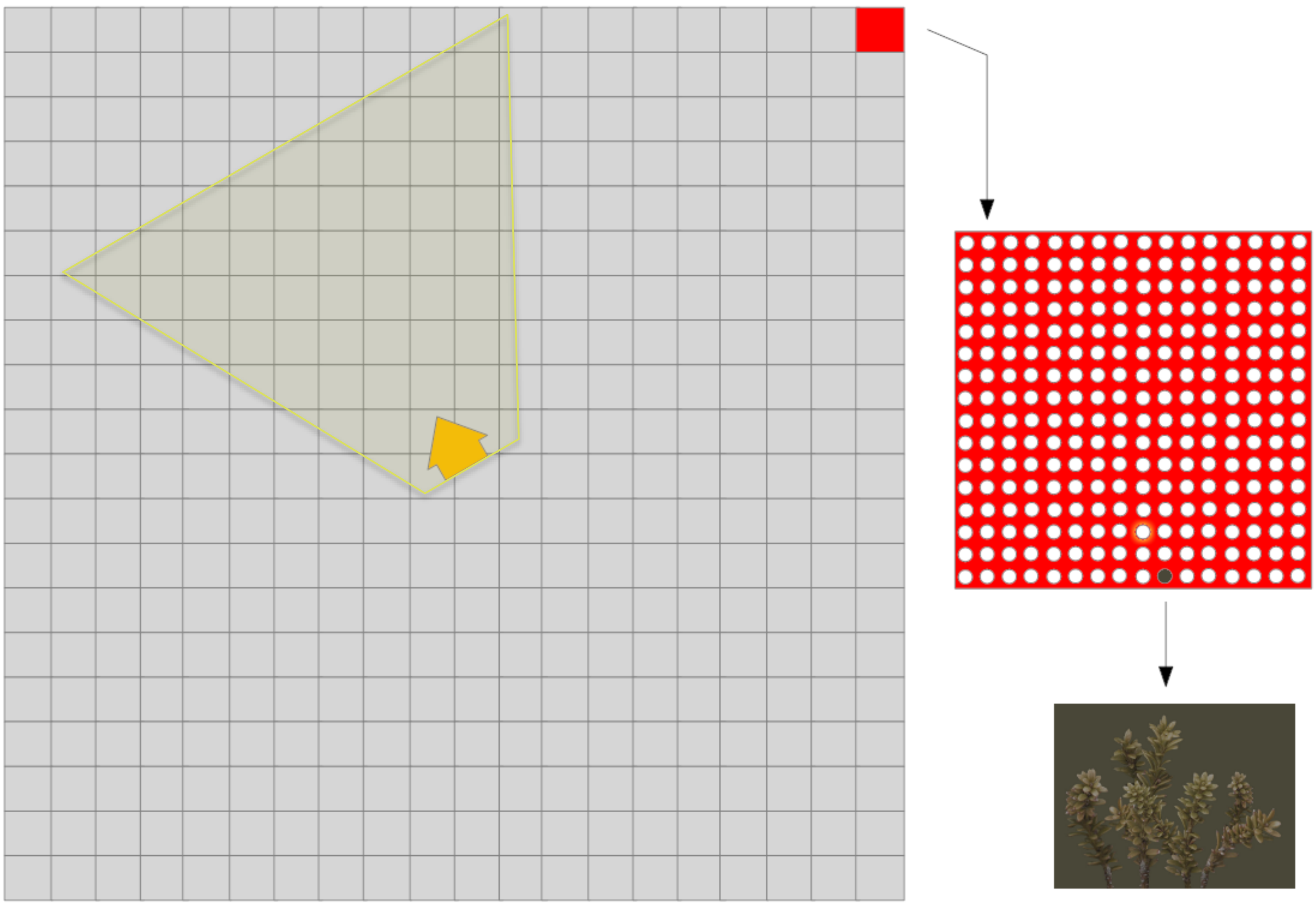
Ground Cover Textures

Ground Cover Rendering Area

Ground Cover Rendering



Ground Cover Processing



Ground Cover Tiles

- Grass rendered in fixed region around camera
- Region divided into 400 tiles of 8 meters square
- Each meter square contains 4 screen aligned sprites
- So each tile contains 256 sprites
  - Each sprite information is encoded as a vector4
- Tiles are cached in memory

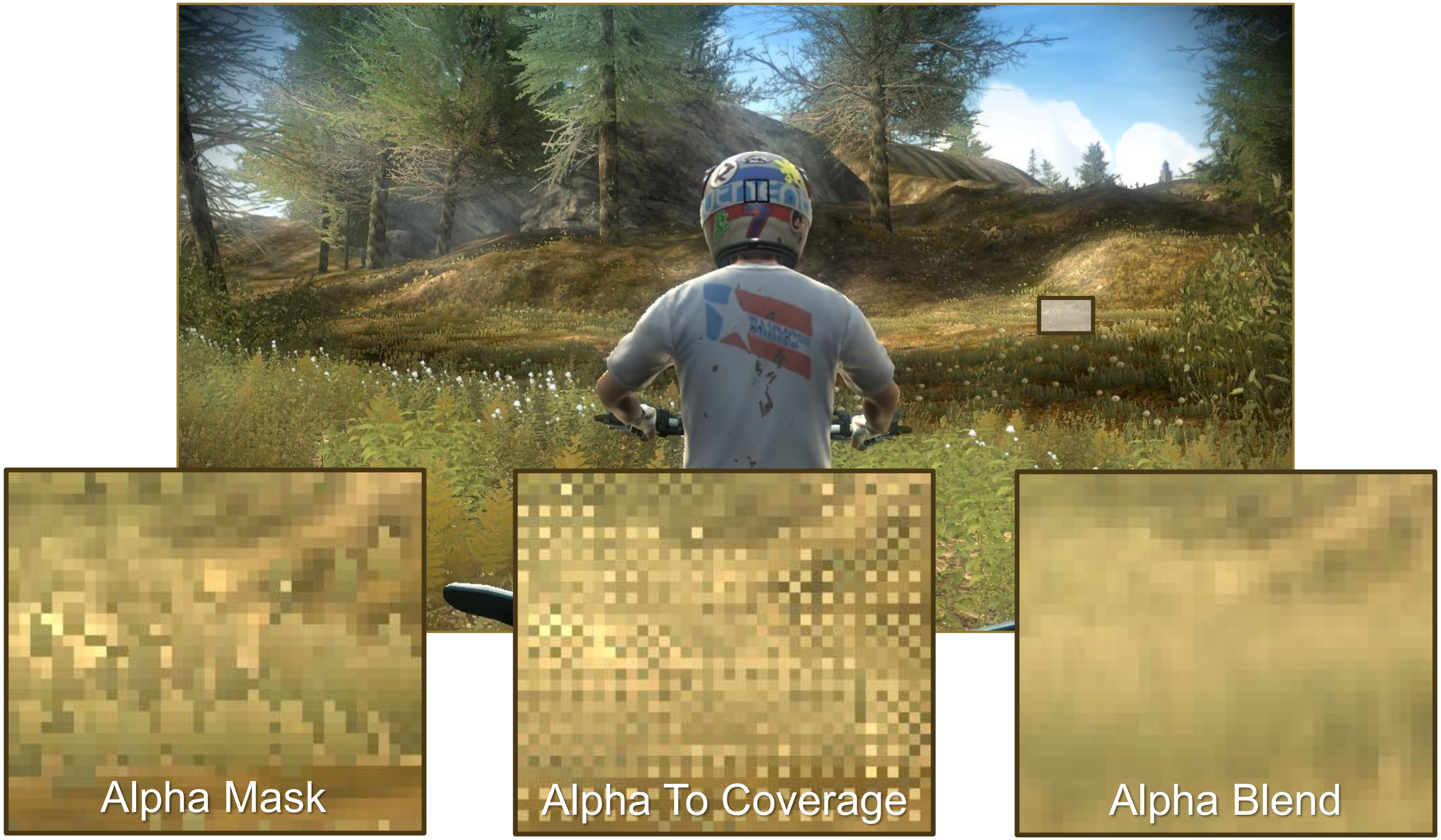
Ground Cover Vertex Buffers

- Vertex buffers are generated per frame
- Tile information is copied from the tile cache
- 16KB needs to be copied by the CPU for each visible tile
- Vertex format and shader details in course notes

Ground Cover Vertex Shader

```
void decode_vertex(INPUT input, out OUTPUT output)
{
    // pos.w integer part contains:
    // corner = grass_type*4 + vertex_index
    float corner = floor( input.pos.w );
    // corner is used to lookup into preset
    // array of uv and size information
    float4 vSpriteInfo = SpriteInfoArray[ corner ];
    // pos.w fractional part contains the random size scale
    // this is halved so that we don't overflow
    float scale = input.pos.w - corner;
    float scale *= 2;
    // fill texture uvs
    output.uv = vSpriteInfo.xy;
    // fill position
    output.pos = float4( input.pos.xyz, 1 );
    // get postion offset from sprite centre
    float2 offset = vSpriteInfo.wz;
    offset *= scale;
    // grow face horizontally
    // up and right are calculated elsewhere in the shader
    output.pos.xz += input.right.xz * offset.x;
    // grow face vertically
    output.pos.xyz += input.up.xyz * offset.y;
}
```

Ground Cover Alpha Compositing



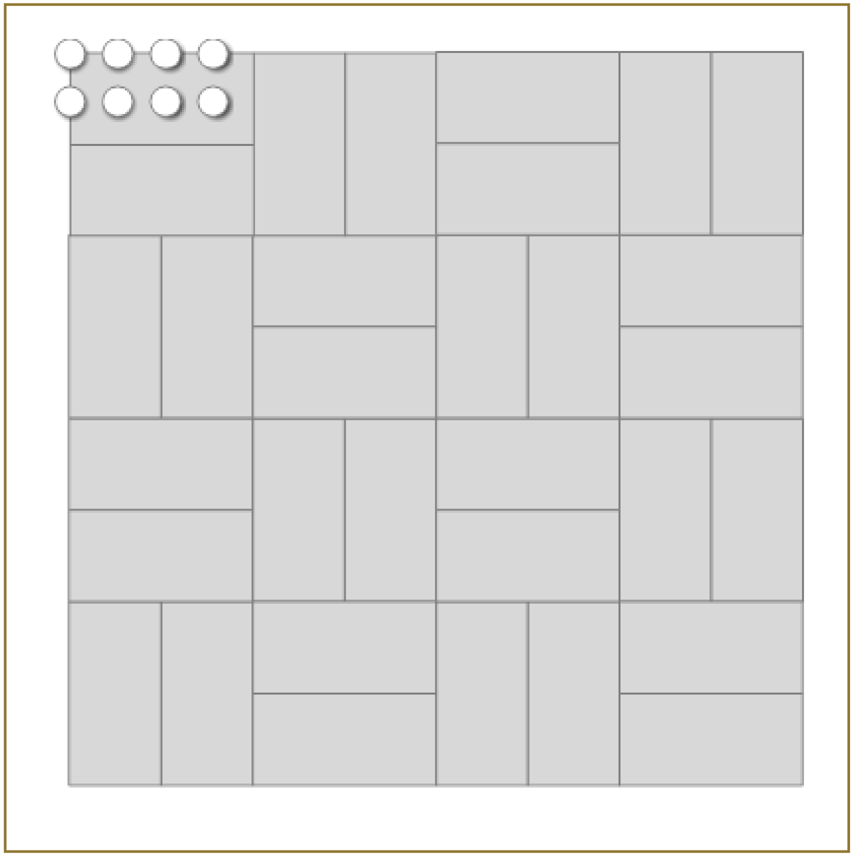
Ground Cover Render Order

- Alpha blending requires geometry sorting
- But regular geometry placement makes sorting easier
- Two levels of granularity
  - Tiles are rendered from back to front
  - Sprites are ordered within each tile

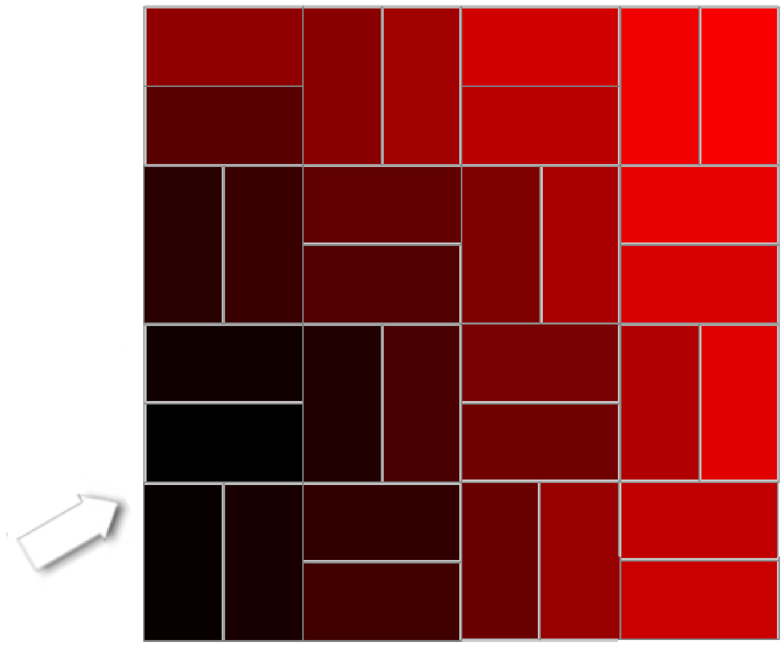
Ground Cover Render Order

- Ordering within each tile is pre-calculated
  - We pre-calculate render orders for 16 camera directions
  - And chose the order for the camera direction that most closely matches the current one
  - For performance we group sprites into 32 “cells” of 8 and only sort at the cell level

Ground Cover Tile Cell Layout



Ground Cover Cell Ordering



Ground Cover Results

Ground Cover Summary

- Advantages
  - High quality alpha blended ground cover
  - Low cost CPU sorting
  - Artist friendly workflow
- Disadvantages
  - High overdraw isn’t cheap for the GPU

Tree Rendering in Pure

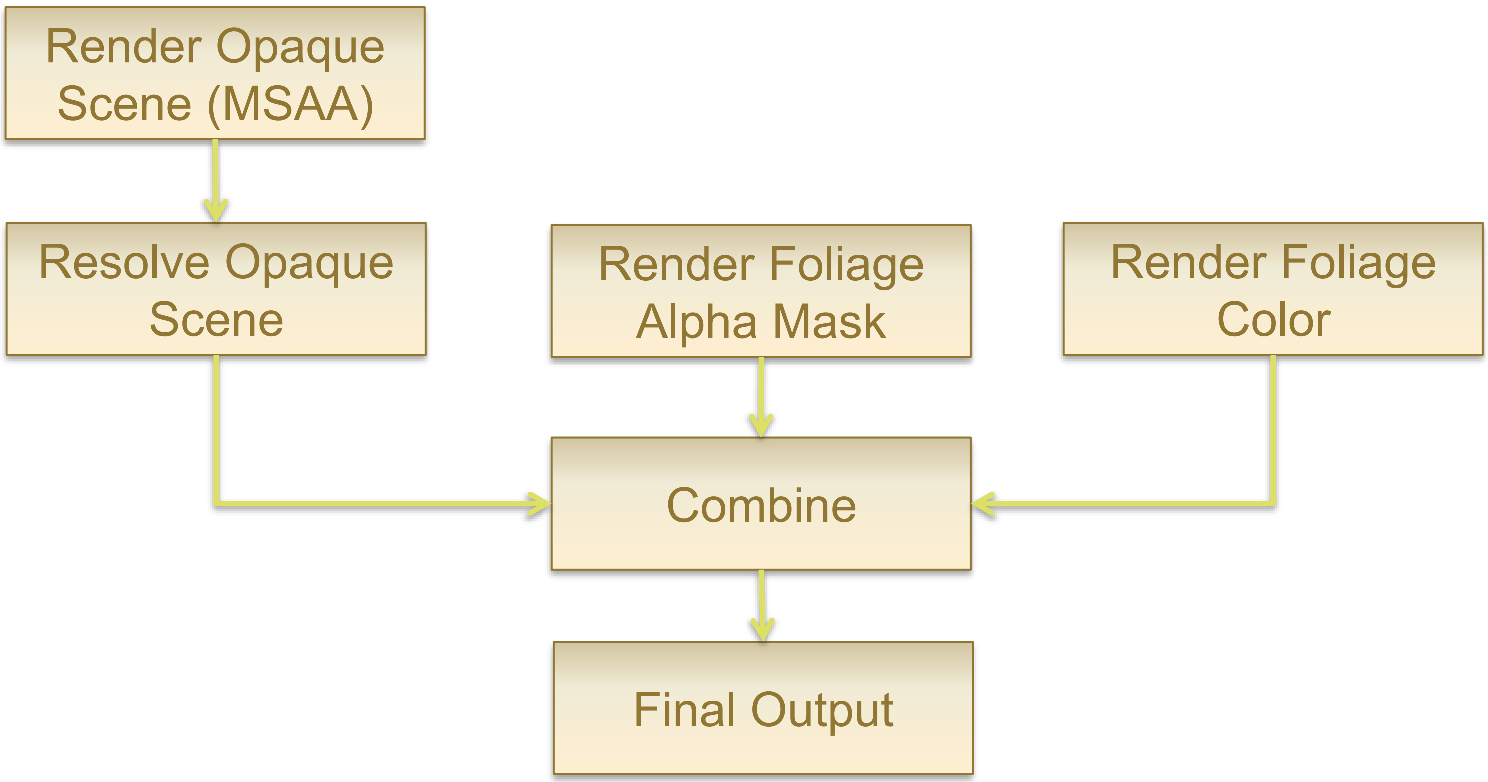
Tree Rendering Challenges

- Trees require more detailed geometry
- Trees rendered further into the distance
- Many trees in the game world, unevenly distributed

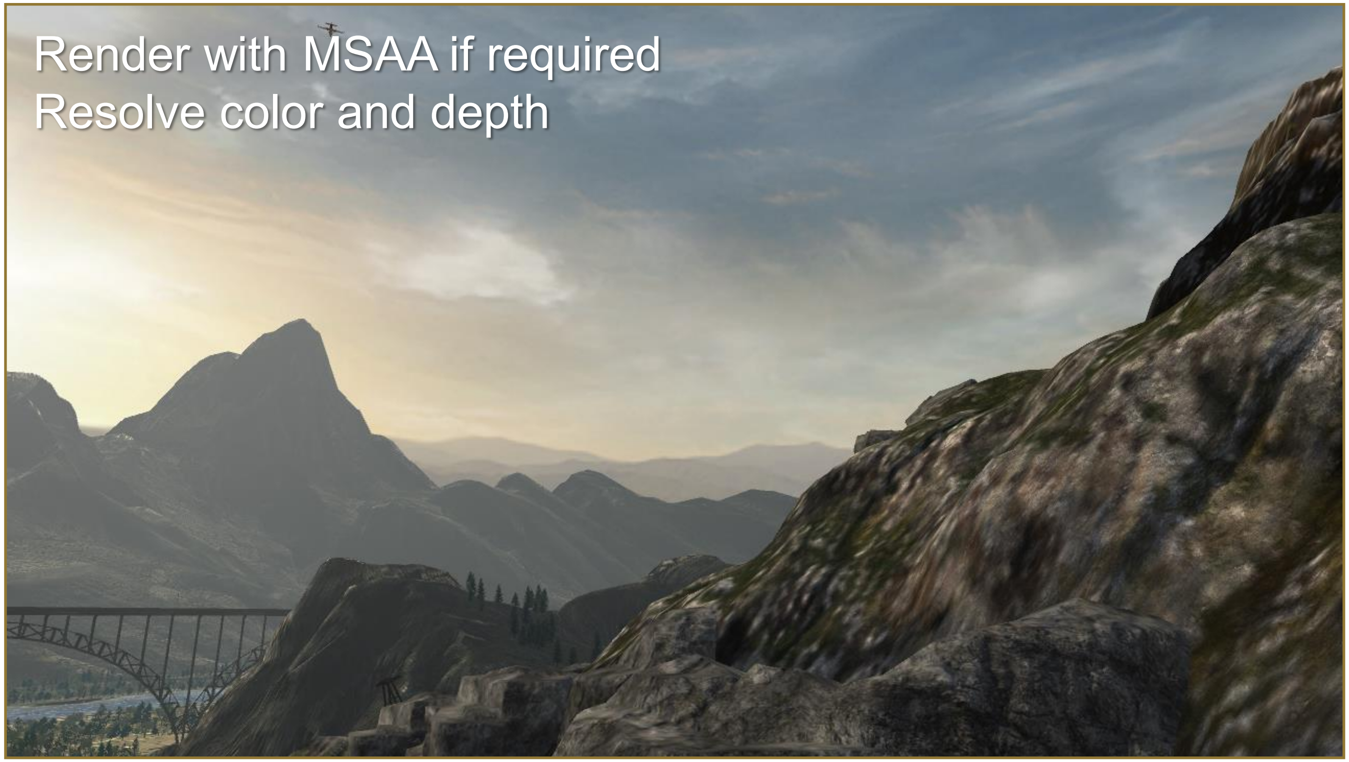
Tree Rendering Observations

- Using alpha blending will require expensive sorting
- Using alpha mask or alpha to coverage doesn't look good enough on its own
- Alpha mask aliasing is only a real problem for the forest silhouette

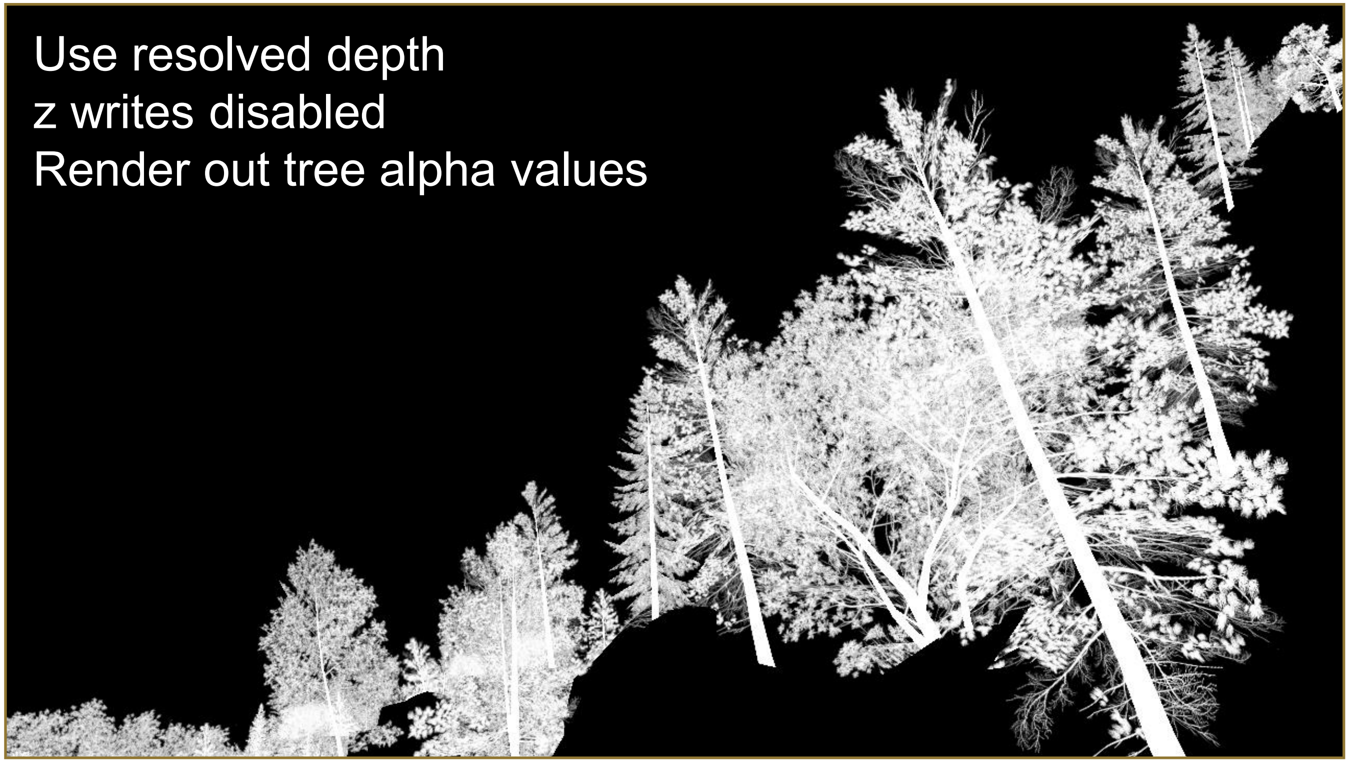
Screen-Space Alpha Mask Overview



Opaque Scene Pass



The Alpha Mask



Alpha Mask Creation Pixel Shader

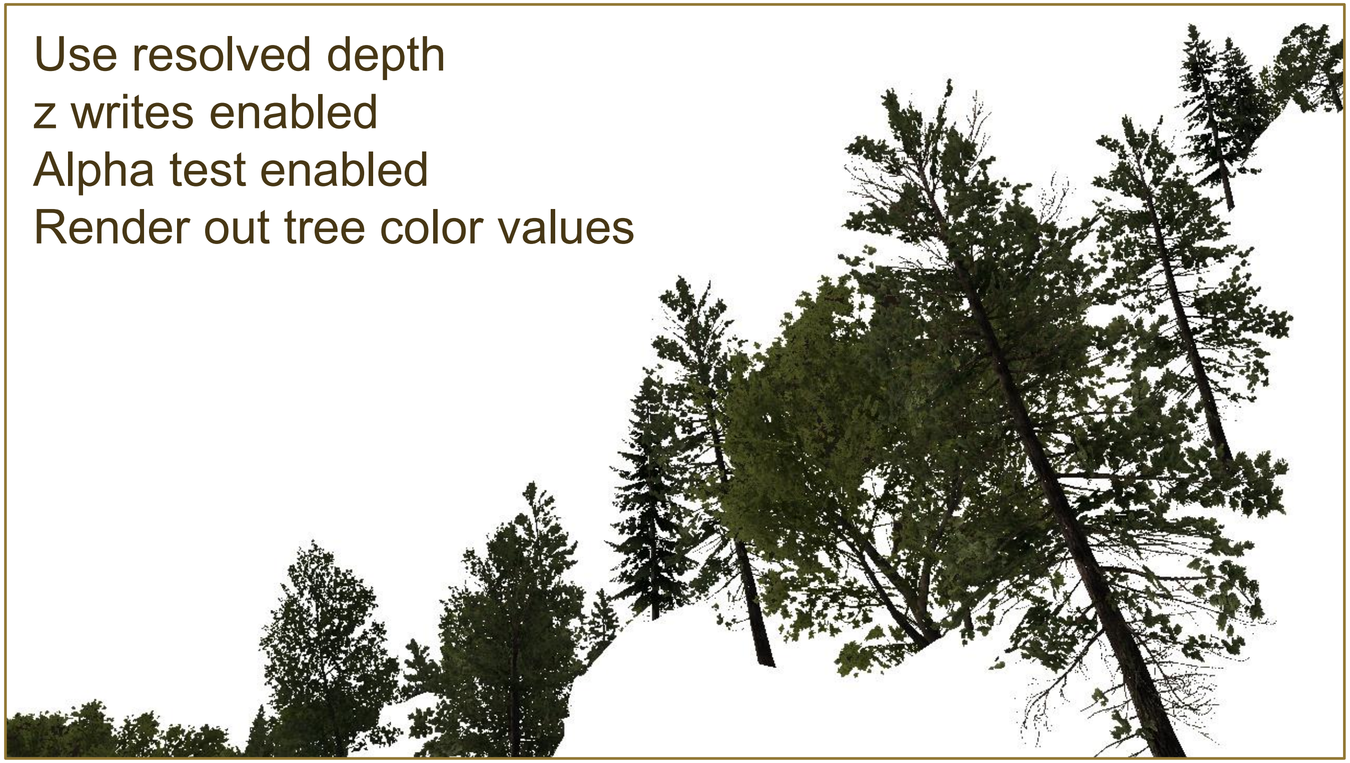
```
sampler alphaTexture : register(s0);

struct PSInput
{
     float2 vTex : TEXCOORD0;
};

float4 main( PSInput In ) : COLOR
{
    return tex2D(  alphaTexture , In.vTex ).aaaa;
}
```

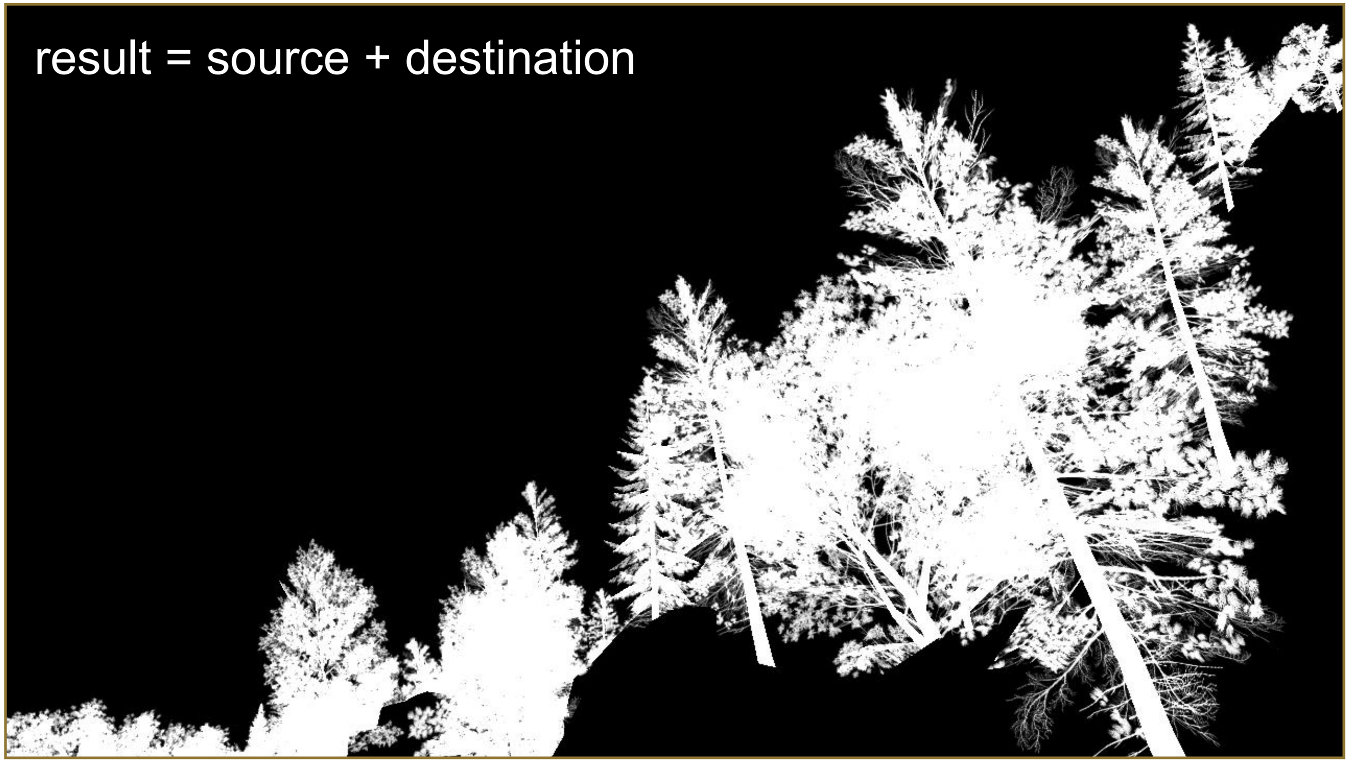
Foliage Color Pass

Use resolved depth
z writes enabled
Alpha test enabled
Render out tree color values

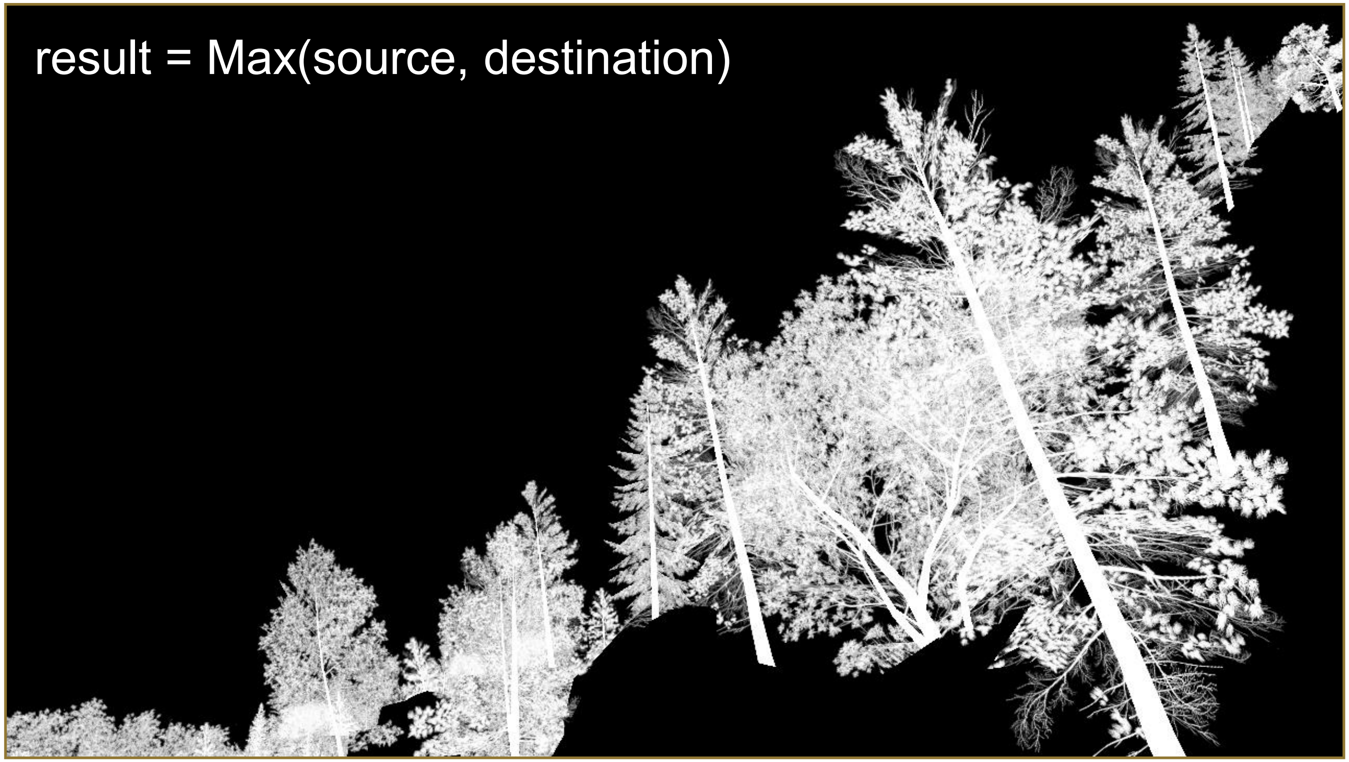


Combine Pass

Alpha Mask Creation using 'ADD'



Alpha Mask Creation using ‘MAX’



Alpha Mask Creation Using Combination

- ADD gives opaque results
- MAX gives more detail and a softer silhouette
- Fortunately we can create both in one pass!
  - Different blend modes for writing to color and alpha components of render target
- Average the two values during final combine pass
- Shader details in course notes

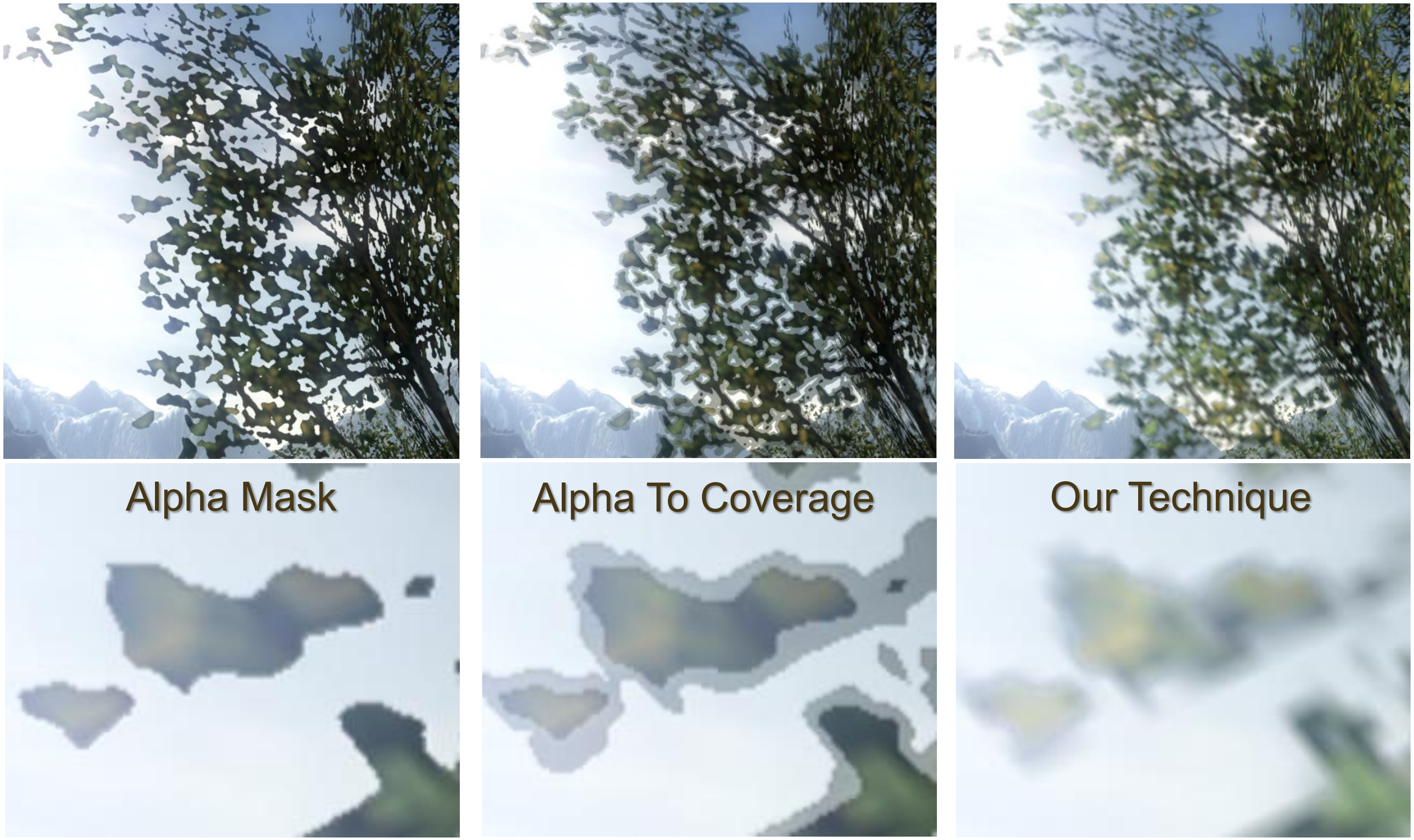
Combine Pixel Shader

```
sampler maskImage: register(s0);
sampler treeImage: register(s1);
sampler worldImage: register(s2);

float4 main( float2 vTexCoord : TEXCOORD ) : COLOR
{
   float4 vTreeTexel = tex2D( treeImage, vTexCoord.xy );
   float4 vWorldTexel = tex2D( worldImage, vTexCoord.xy );
   float4 vMaskTexel = tex2D( maskImage, vTexCoord.xy );

   float lerpValue = (vMaskTexel.r +vMmaskTexel.a) * 0.5f;
   return lerp( vWorldTexel, vTreeTexel, lerpValue );
}
```

Comparison With Simple Compositing



The Final Result

Summary

- Advantages
  - High quality anti-aliased silhouettes
  - No MSAA used for alpha rendering
  - No sorting required
- Disadvantages
  - Cost of additional render passes
  - Works best when internal aliasing isn't a problem

Part 2: Split/Second

Split/Second

- Racing game for Xbox360, PlayStation3 and PC
- Release Q2 2010
- Arcade racing game

Art Direction

Art Direction

This Talk

- Overview of some of the render techniques we use
  - Deferred Shading
  - Deferred Shadowing
  - Irradiance Volumes
- Some specific optimizations we've employed along the way

Video of Split/Second

Deferred Shading

- Split/Second uses a deferred shading renderer

Deferred Shading

- Decouples lighting from geometry
- Information needed for lighting is written to a Geometric Buffer (G-Buffer) as main scene is rendered
- The lighting of the scene is deferred until the lighting pass which happens during the post-processing phase

Deferred Shading

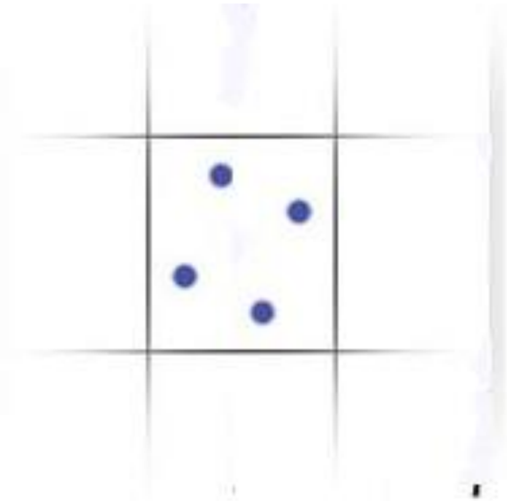
Render Target	Channel 1	Channel 2	Channel 3	Channel 4
1	Albedo.r	Albedo .b	Albedo.g	Unlit.r
2	Normal.x	Normal.y	Normal.z	Unlit.b & Edge
3	Unlit.g	Specular	Motion.x	Motion.y

Optimisation

- 1st Optimisation is for MSAA

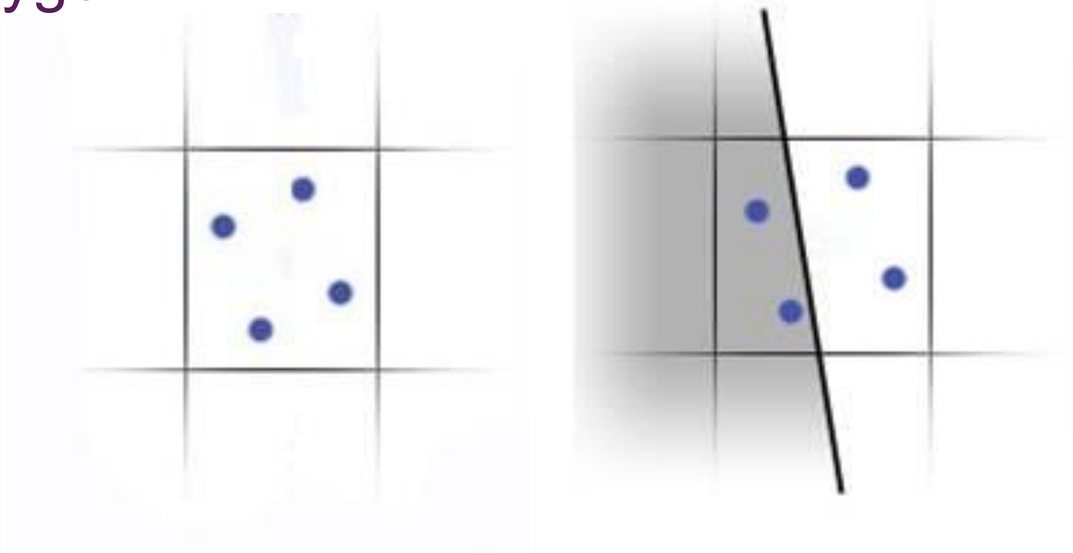
Multi-Sample Anti-Aliasing

- Variation of Full Screen Anti Aliasing
- FSAA renders scene to a higher resolution than required
- The averages down to the desired resolution
- This has serious performance implications



Multi-Sample Anti-Aliasing

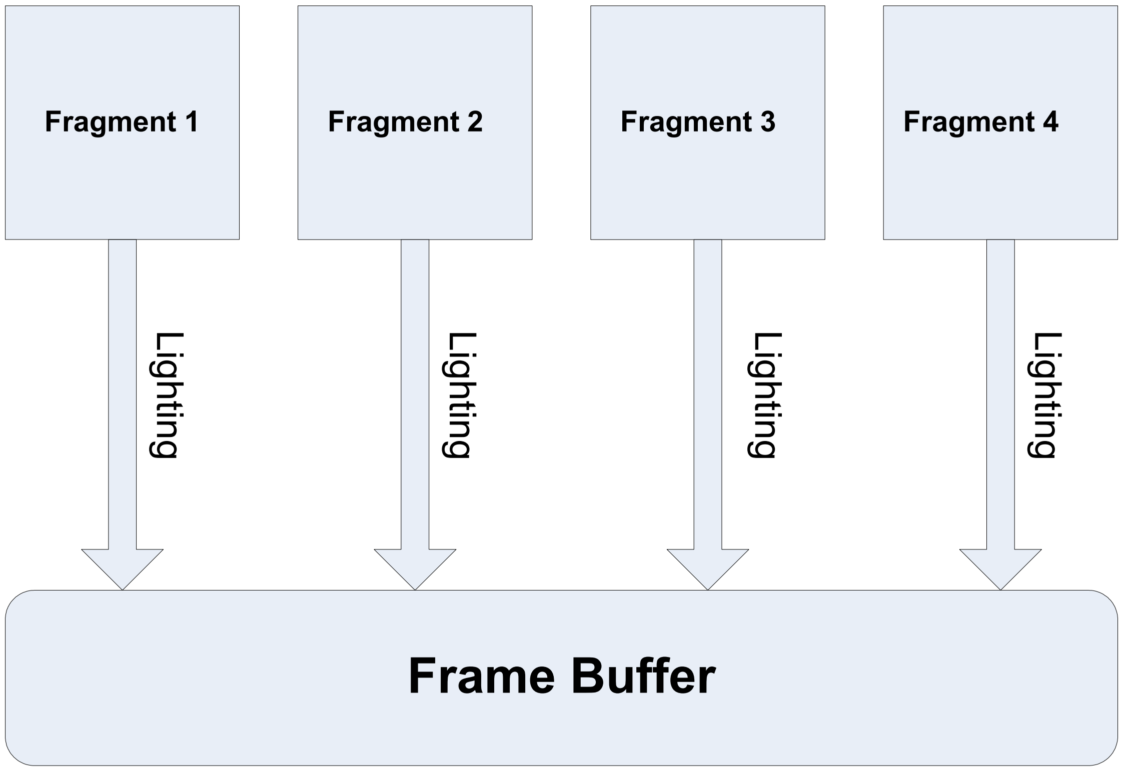
- MSAA runs the pixel shader only once per pixel
- Sets the fragment color for each fragment covered by the polygon



Multi-Sample Anti-Aliasing

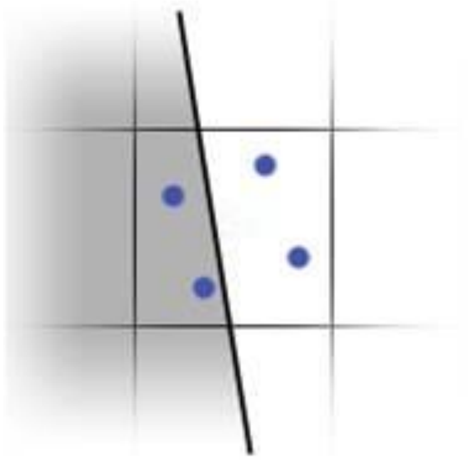
- Can't use hardware to average the fragments because the G-Buffer is not suitable for interpolation
- This means we have to manually blend every fragment

Multi-Sample Anti-Aliasing

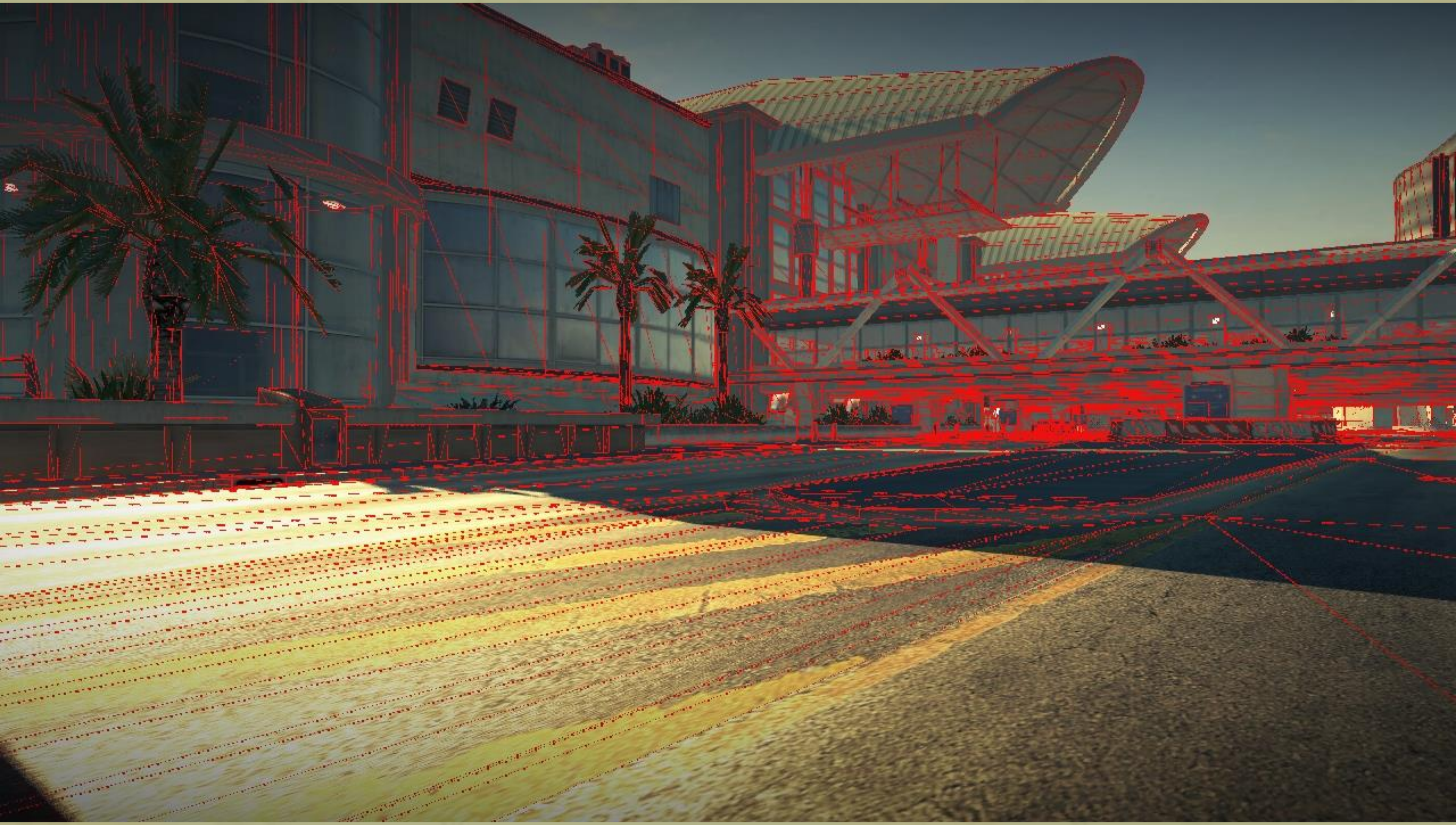


Multi-Sample Anti-Aliasing

- We observe that 85% of the pixels are interior to a polygon
- This means all their fragments are identical
- Can we quickly identify the 15% which are different?

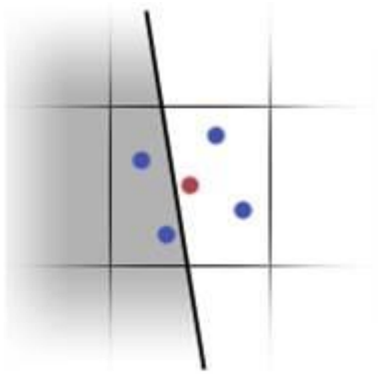


Fragments that need to be identified



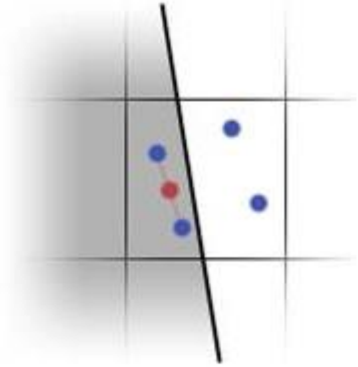
Centroid Sampling

- We use a piece of hardware which is also trying to identify polygon edges
- Centroid sampling avoids vertex attributes being sampled beyond the polygon's boundaries



Centroid Sampling

- Centroid sampling adjusts the position used for determining colour to be the centre of all the sampling points covered by the polygon
- So if the centroid moves then we're on a polygon edge



Centroid Sampling

- Fortunately we interrogate the value of the centroid sample in the pixel shader
- If it’s not zero we know the triangle doesn’t cover all of the samples for the pixel

Centroid Sampling

```
struct PSInput
{
    float4 vPos : TEXCOORD0;
    float4 vPosCentroid : TEXCOORD1_CENTROID;
};

float4 main( PSInput In ) : COLOR
{
    float2 vEdge = In.vPosCentroid.xy - In.vPos.xy;
    float fEdge = (vEdge.x + vEdge.y == 0.0f) ? 0.0f : 1.0f;

    // For deferred shading we would usually pack this value into
    // one bit in the G-Buffer.
    return float4( fEdge );
}
```

Performance Stats (Xbox360)

Light both fragments	4.5ms
Light fragment 1	2.3ms
Light fragment 2	0.7ms

Deferred Shadowing

- We use parallel split shadow maps for sunlight shadows
- Using the shadow maps and depth buffer we create a screen space shadow mask
- This is then used in the lighting pass to attenuate the lighting

Shadow Maps

Zoomed In

Shadow Edge Detection

- To avoid these artifacts we use percentage closer filtering
- PCF takes multiple samples from the shadow map
- Performs depth test with each of them against a shadow receiver
- Then averages the results.

Shadow Maps with PCF

Zoomed In

Percentage Closer Filtering

- PCF is expensive

Optimisation 2

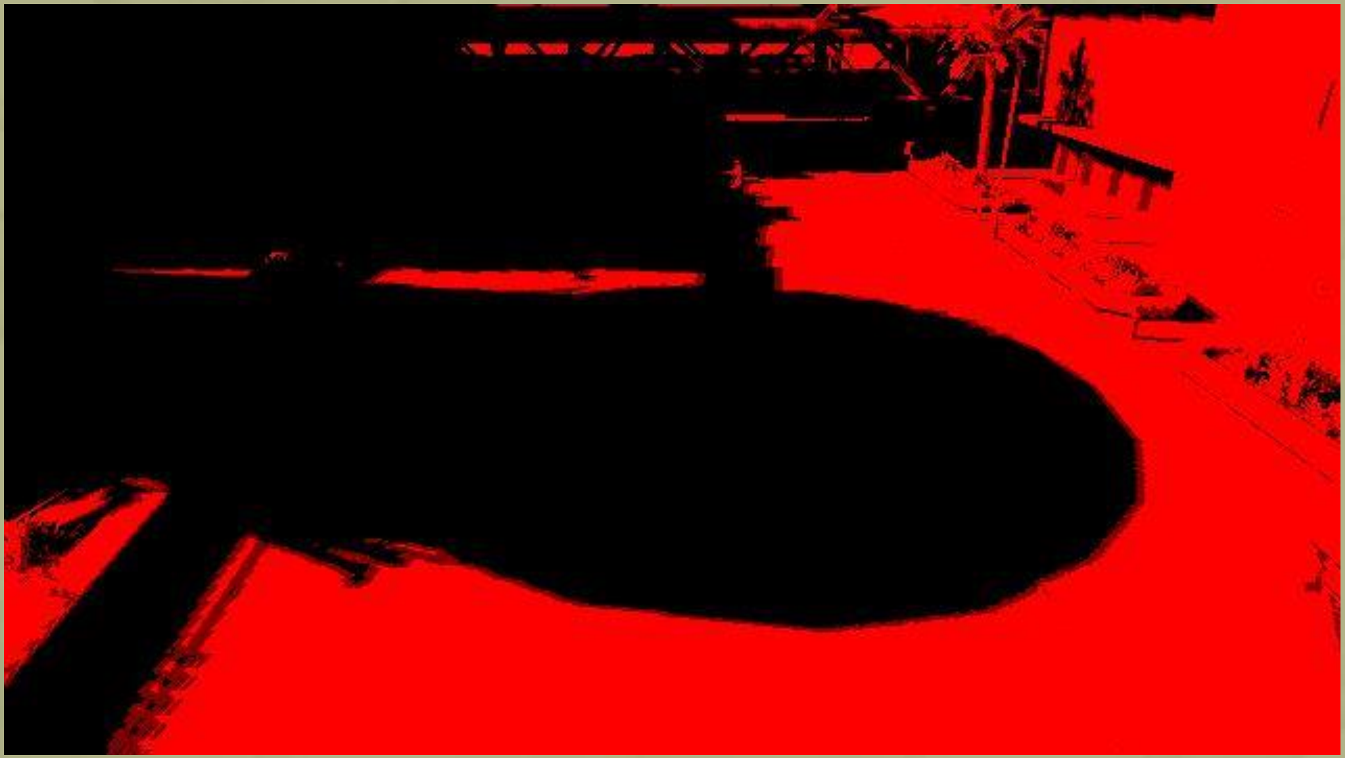
- If we can divide the screen into 3 areas
  - Areas definitely in shadow
  - Areas definitely not in shadow
  - Areas that may be in or out of shadow

- We only want to apply PCF to the last of these areas

Areas that may be in or out of shadow

- Can we work them out exactly
- No. But we can approximate
- We want to generate a mask to show where the PCF must be performed

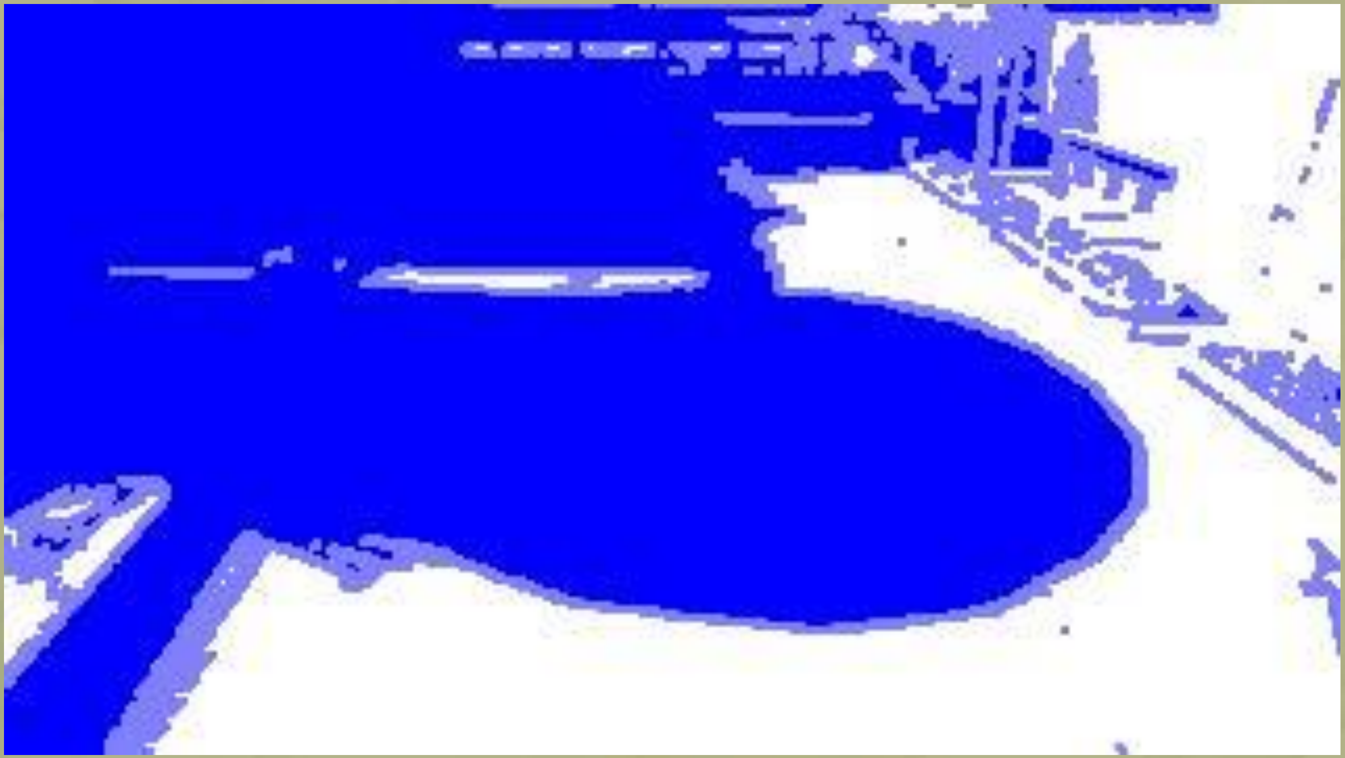
¼ size (cheap)



2nd Pass

- We then perform a second pass
- This time at 1/16th screen size
- During this pass we expand the edges using a conservative algorithm

This is what we end up with



Performance (Xbox360)

- This mask is then fed into the shadow renderer
- Very efficient because it's 1/16th screen size so doesn't use much texture bandwidth

Without mask	6.4ms
With mask	1.7ms
Calc mask	0.6ms

Irradiance Volumes

- We want global illumination in a way that integrates well with our deferred shading pipeline
- Environment changes dramatically in Split/Second so we needed a solution that can respond to those changes
- We thought Irradiance Volumes provided right balance between approximation of GI and ability to update

But…..

- Real-time update is not something we currently do
- This whole feature is WIP

Irradiance Volumes

- IV is a 3D map of diffuse lighting samples
- Irradiance environment maps can be compactly represented in terms of Spherical Harmonics

Probe Placement

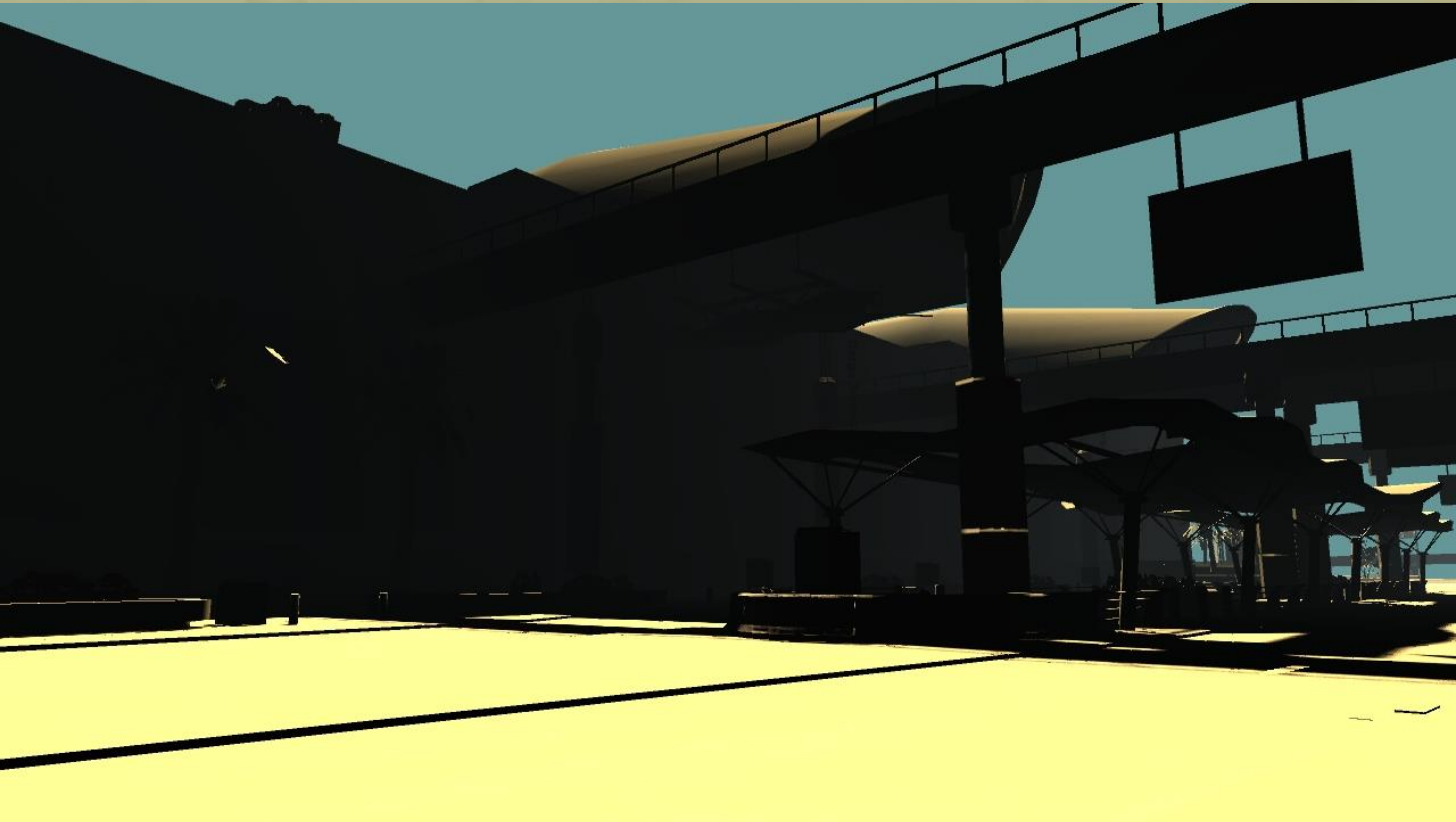
Irradiance Volumes

- With a forward renderer SH coefficients would be uploaded with each render batch
- In S/S we apply the irradiance contribution in the deferred lighting pass
- Advantage: We only pay the calculation cost once per screen pixel

Irradiance Volumes

- We use Volume Textures
- These map naturally to the GPU
- The GPU carries out tri-linear interpolation of the coefficients to give smooth representation

Direct Lighting



Indirect Lighting

Direct + Indirect Lighting

Constant Ambient

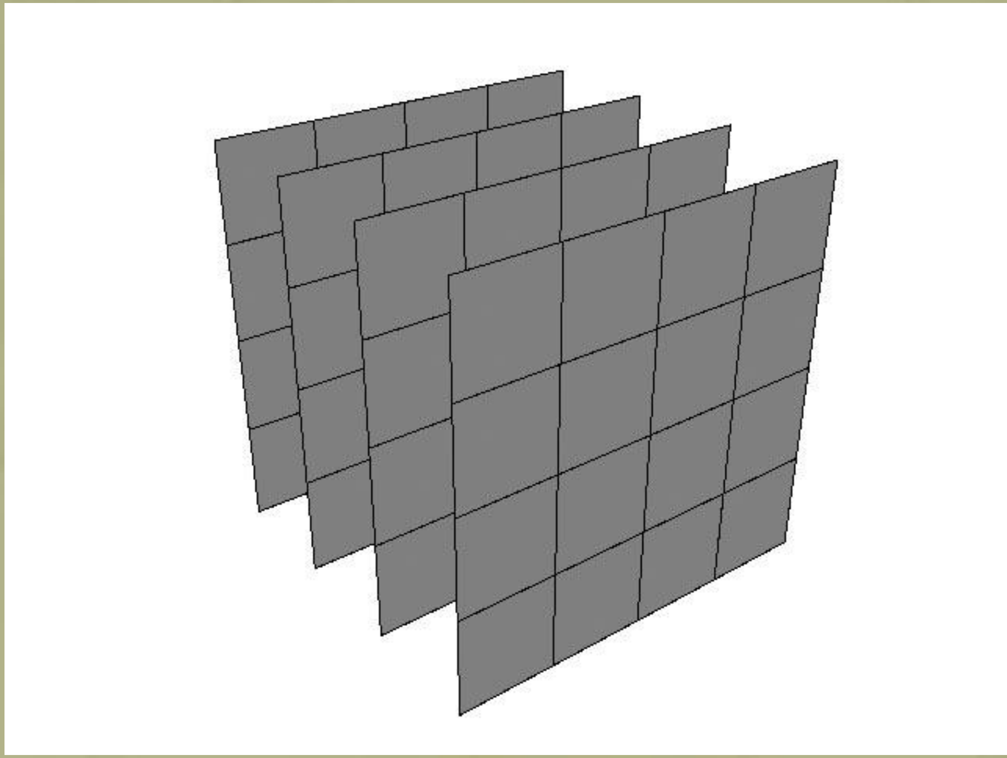
Problems to Solve

- Draw Distance
- Sampling Cost

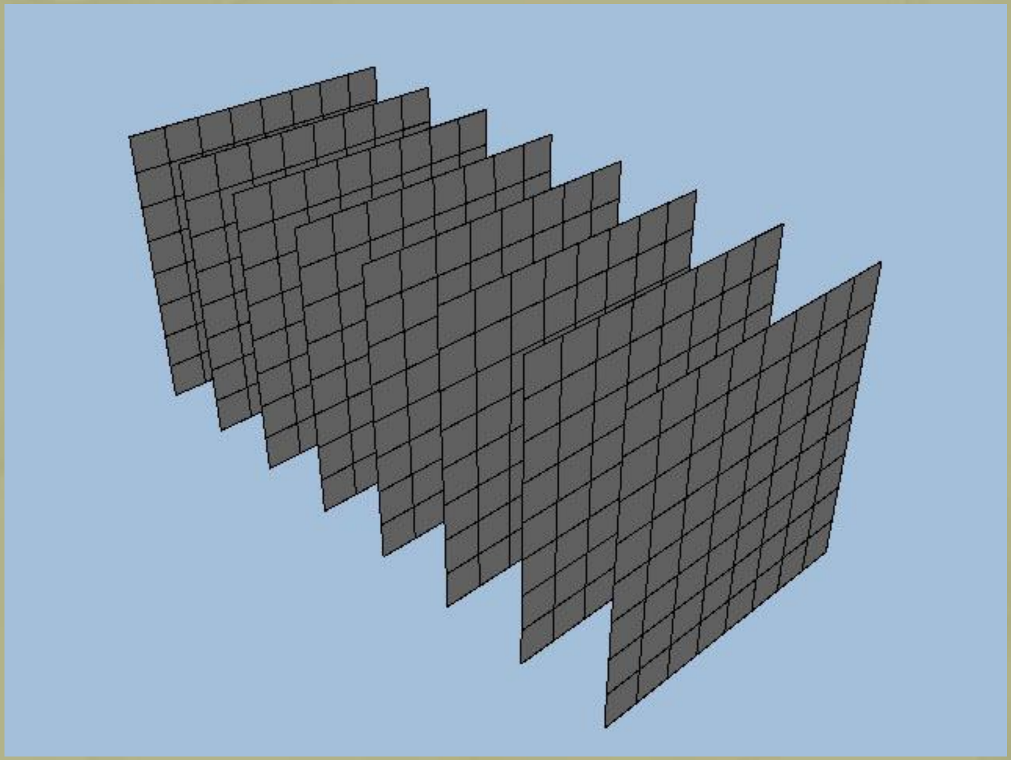
Draw Distance

- Can’t cover the whole scene
- Only cover the area around the camera
- But then what do you do at the border?

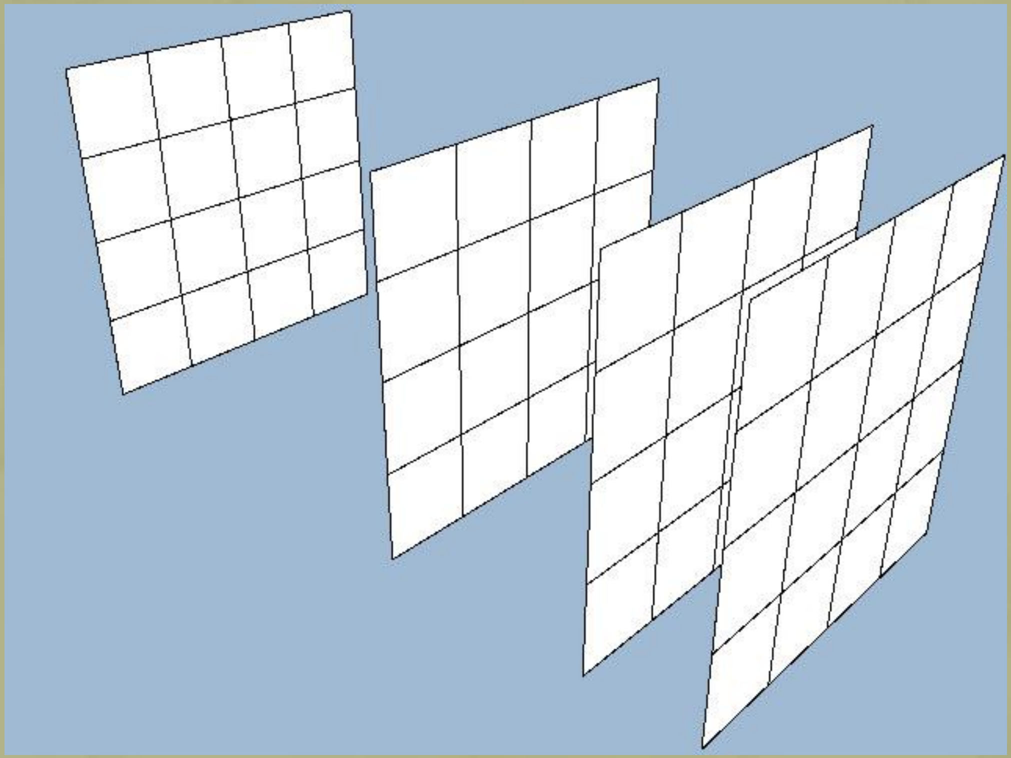
Volume Texture



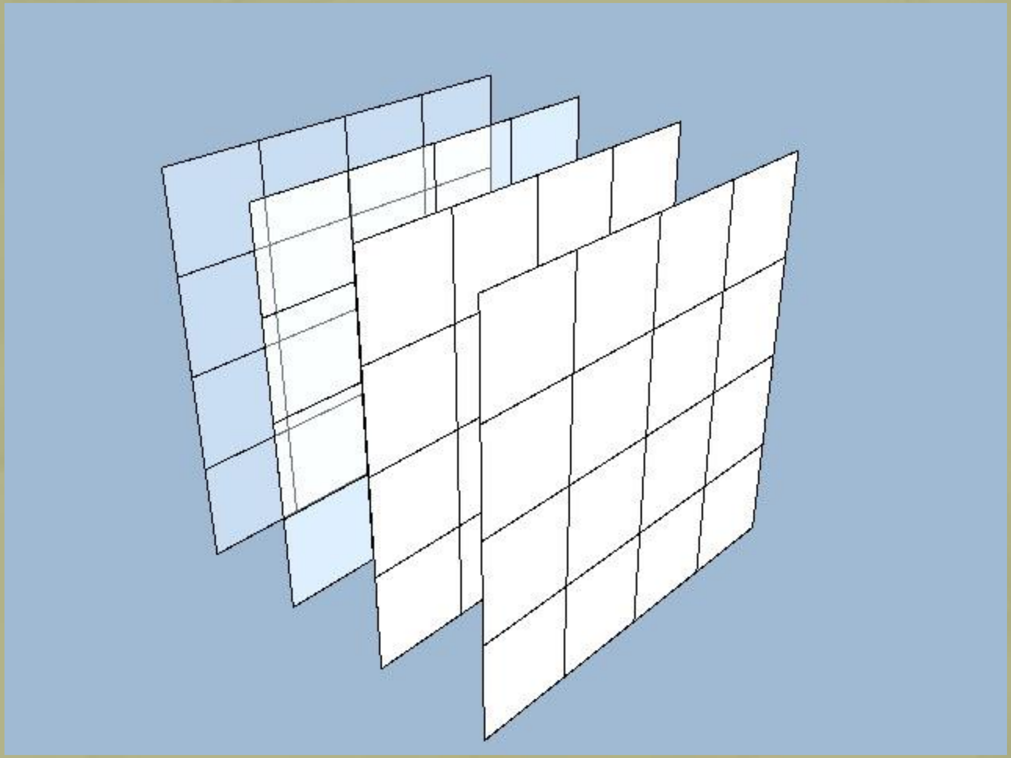
Increased Size



Non-Linear Scale



Fixed Value Interpolation



Shading Cost

Representation	Coefficients per Color Channel	Volume Textures Required	Cost
2nd order spherical harmonics	9	7	8.8 ms
1st order spherical harmonics	4	3	4.6 ms

Shading Cost

- Even 4.6ms would be prohibitively expensive for game
- Explored the optimisation of rendering the indirect lighting to a ¼ sized buffer
- This reduces 2^{nd} order to acceptable 2.2ms

Quarter Sized Indirect Lighting

Quarter Sized Indirect Lighting

Summary

- The deferred shading system has changed the way we optimise. It’s all about screen space now.
- We’ve got more work to do to find out how to use irradiance volumes most effectively.

Acknowledgements

- The Pure team, especially Ben Hathaway, George Parish, Damyan Pepper and James Callin
- The Split/Second team, especially Matt Ritchie and Balor Knight who developed the techniques shown in this presentation

Questions?

- Jeremy.Moore@disney.com
- David.Jefferies@disney.com

References

- [BAVOILMYERS08] BAVOIL, L., AND MYERS, K. 2008. Deferred Rendering using a Stencil Routed K-Buffer, *ShaderX^6,* Engel, W. (Editor), Charles River Media.
- [EVERITT01] EVERITT, C. 2001. Interactive Order-Independent Transparency.
- *[KCS07] KHARLAMOV, A., CANTLAY, I., AND STEPANENKO, Y. 2007. Next-Generation SpeedTree Rendering, GPU Gems 3,* Nguyen, H. (Editor), Addison-Wesley.
- [PORTERDUFF84] PORTER, T., AND DUFF, T. 1984. *Compositing Digital Images. Computer Graphics* 18, 3, pp 253-259.

References

- [ENGEL09] ENGEL, W. 2009. Designing a Renderer for Multiple Lights: The Light Pre-Pass Renderer, *ShaderX⁷*, Engel, W. (Editor), Charles River Media.
- [GSHG98] GREGER, G., SHIRLEY, P., HUBBARD, P. M., AND GREENBERG, D.P. 1998. The Irradiance Volume, *IEEE Computer Graphics & Applications*, 18, 2, pp. 32-43.
- [HARGREAVES04] HARGREAVES, S. 2004. Deferred Shading, *Game Developers Conference*, D3D Tutorial Day, March, 2004.
- [KOONE07], KOONE, R. 2007. Deferred Shading in Tabula Rasa, *GPU Gems 3*, Nguyen, H. (Editor), Addison-Wesley.
- [MMG06] MITCHELL, J. L., MCTAGGART, G., AND GREEN, C. 2006. Shading in Valve's Source Engine, *Advanced Real-Time Rendering in 3D Graphics and Games - SIGGRAPH 2006*, pp. 129-142.
- [MITTRING07] MITTRING, M. 2007. Finding Next Gen – CryEngine 2.0, *Advanced Real-Time Rendering in 3D Graphics and Games - SIGGRAPH 2007*, pp. 97-121.

References

- [OAT07] OAT, C. 2007. Irradiance Volumes for Real-Time Rendering, *ShaderX*[5]*,* Engel, W. (Editor), Charles River Media.
- [RAMAMOORTHIHANRAHAN01] RAMAMOORTHI, R., AND HANRAHAN, P. 2001. An Efficient Representation for Irradiance Environment Maps, *Proceedings of ACM SIGGRAPH 2001*, pp. 497–500.
- [SHISHKOVTSOV05], SHISHKOVTSOV, O. 2005. Deferred Shading in S.T.A.L.K.E.R., *GPU Gems 2*, Pharr, M., Fernando, R. (Editors), Addison-Wesley.
- [SLOAN04] SLOAN, P.-P. 2004. Efficient Evaluation of Irradiance Environment Maps, *ShaderX*[2]*,* Engel, W. (Editor), Charles River Media.
- [ZSXL06] ZHANG, F., SUN, H., XU, L., AND LUN, L. K. 2006. Parallel-Split Shadow Maps for Large-Scale Virtual Environments, *Proceedings of ACM International Conference on Virtual Reality Continuum and Its Applications 2006*, pp. 311–318.