Rendering

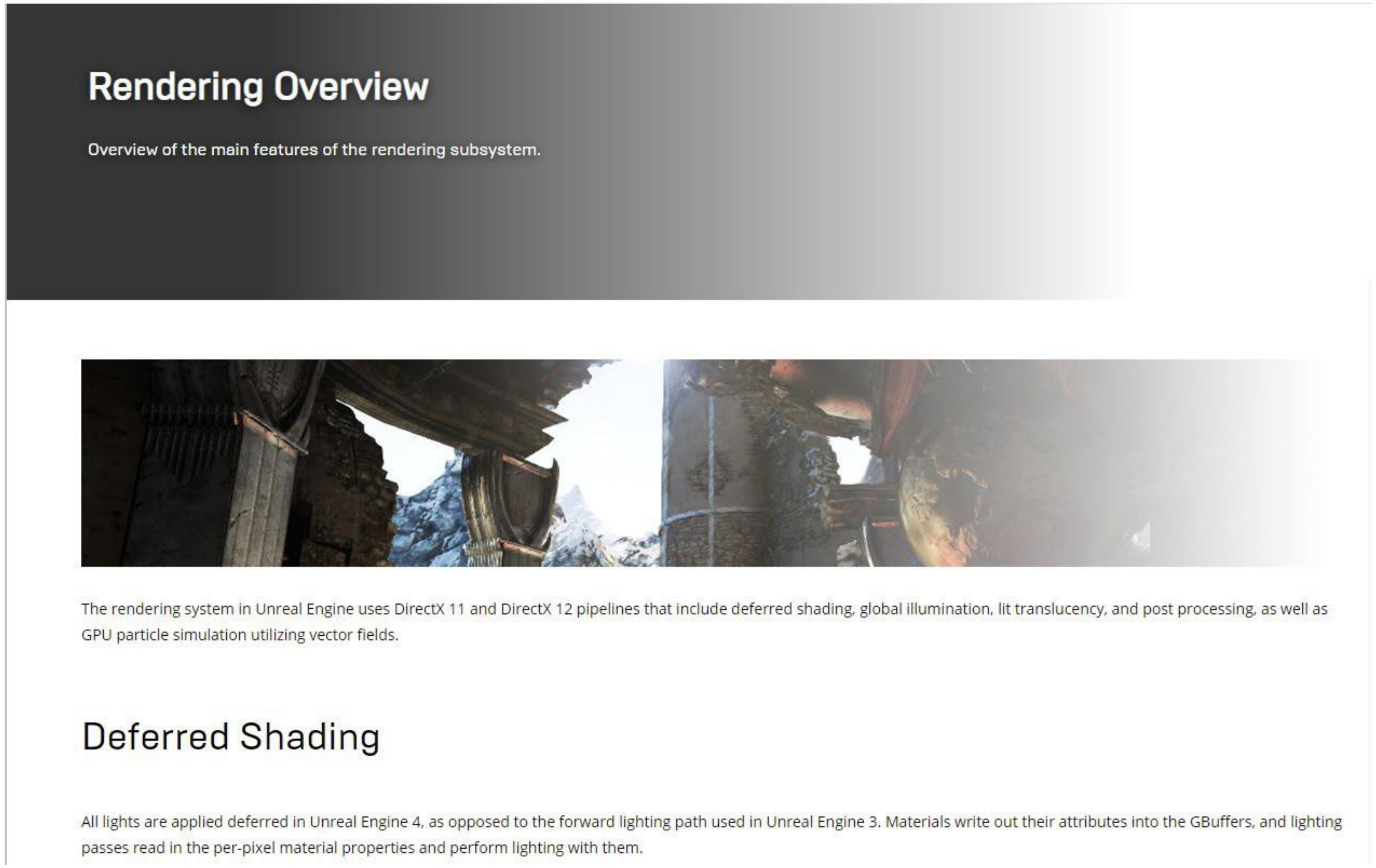
Deferred Shading

CS 415: Game Development

Professor Eric Shaffer

ILLINOIS

UE4 Uses Deferred Shading



Rendering Overview

Overview of the main features of the rendering subsystem.

The rendering system in Unreal Engine uses DirectX 11 and DirectX 12 pipelines that include deferred shading, global illumination, lit translucency, and post processing, as well as GPU particle simulation utilizing vector fields.

Deferred Shading

All lights are applied deferred in Unreal Engine 4, as opposed to the forward lighting path used in Unreal Engine 3. Materials write out their attributes into the GBuffers, and lighting passes read in the per-pixel material properties and perform lighting with them.

Today we'll answer:

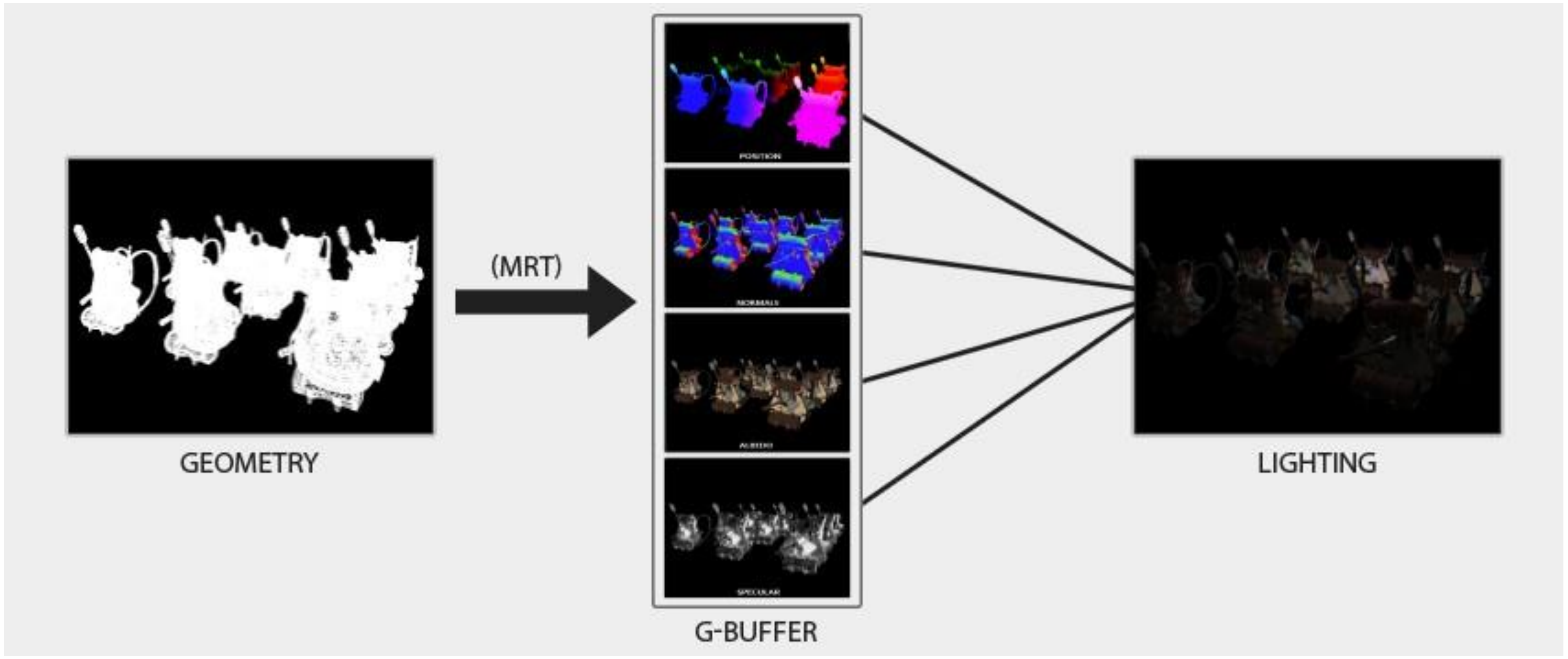
What is it?

Why would you use it?

Why wouldn't you use it?

Deferred Shading: Basic Idea

- Don’t bother with any lighting while drawing scene geometry
- Use multiple buffers to store data such as the position and normal of each pixel
- Apply lighting as a 2D postprocess, using these buffers as input



Comparison: Single Pass Lighting

```
For each object:
  Render mesh, applying all lights in one shader
```

- Good for scenes with small numbers of lights (eg. outdoor sunlight)
- Difficult to organize if there are many lights

Comparison: Multipass Lighting

```
For each light:
   For each object affected by the light:
      framebuffer += object * light
```

- Worst case complexity is *num_objects * num_lights*
- Sorting by light or by object are mutually exclusive: hard to maintain good batching
- Ideally the scene should be split exactly along light boundaries, but getting this right for dynamic lights can be a lot of CPU work

Deferred Shading

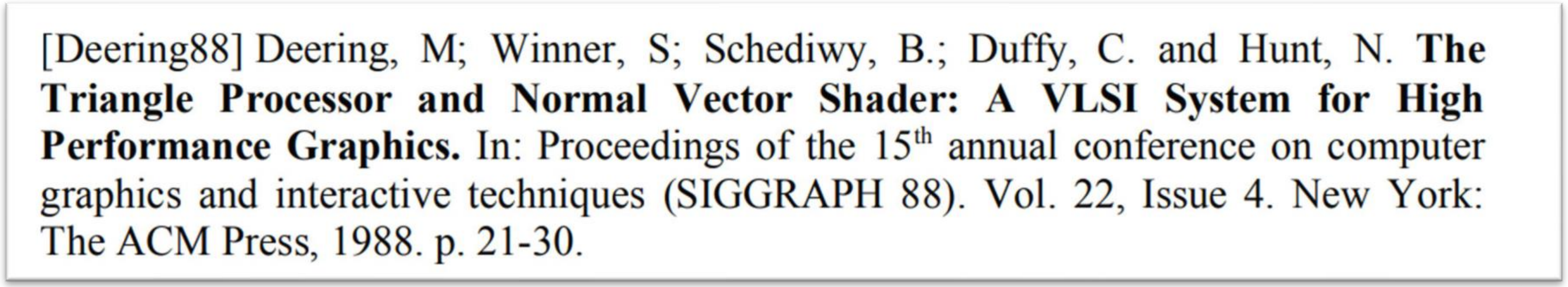
```
For each object:
  Render to multiple targets

For each light:
  Apply light as a 2D postprocess
```

- Worst case complexity is *num_objects + num_lights*
- Perfect batching
- Many small lights are just as cheap as a few big ones

Not a New Idea

The concept was first introduced in a hardware architecture in 1988



[Deering88] Deering, M; Winner, S; Schediwy, B.; Duffy, C. and Hunt, N. **The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics.** In: Proceedings of the 15th annual conference on computer graphics and interactive techniques (SIGGRAPH 88). Vol. 22, Issue 4. New York: The ACM Press, 1988. p. 21-30.

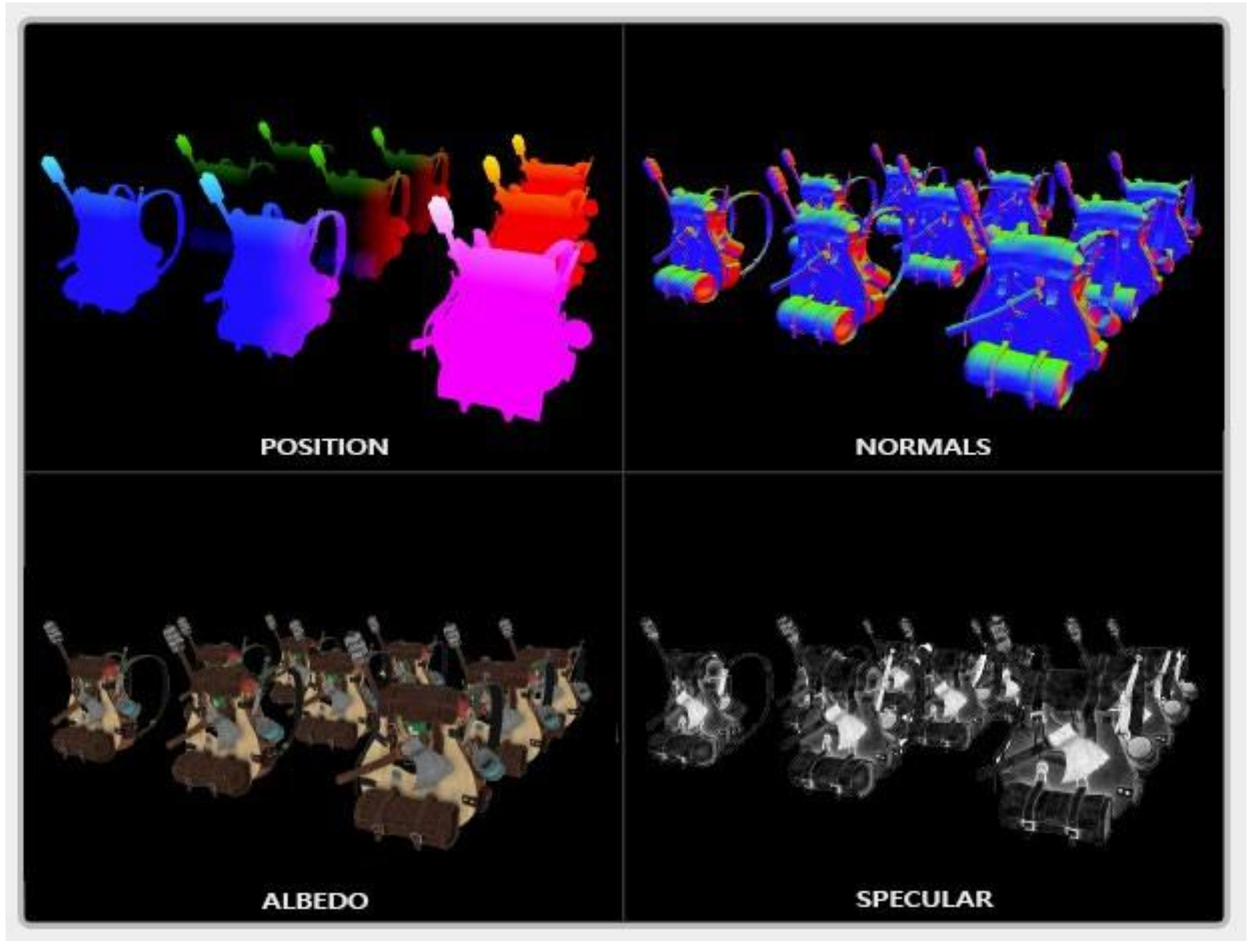
Later included as a part of the experimental PixelFlow system

Multiple Render Targets

- Required outputs from the geometry rendering are:
  - Position
  - Normal
  - Material parameters (diffuse color, emissive, specular amount, specular power, etc)

- 3 or 4 render targets (buffers) is reasonable
- More is not...this is not good if your lighting needs many input values!

G-Buffer: Geometry Buffer



4 Buffers (4 render targets)

Each contains a frame of pixels

…they are textures

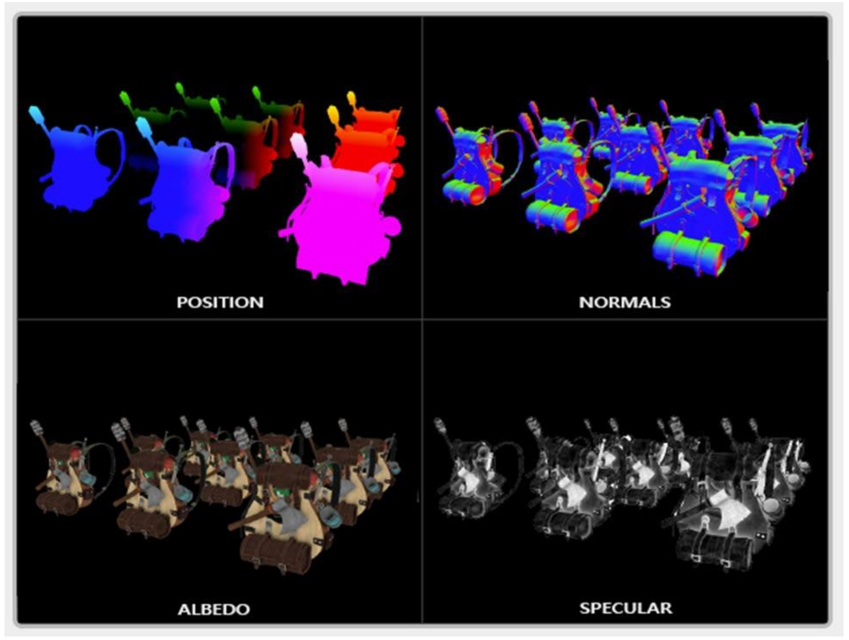
Each pixel contains a piece of information needed for shading

Might not be divided up this way…some systems use different number of buffers/different data

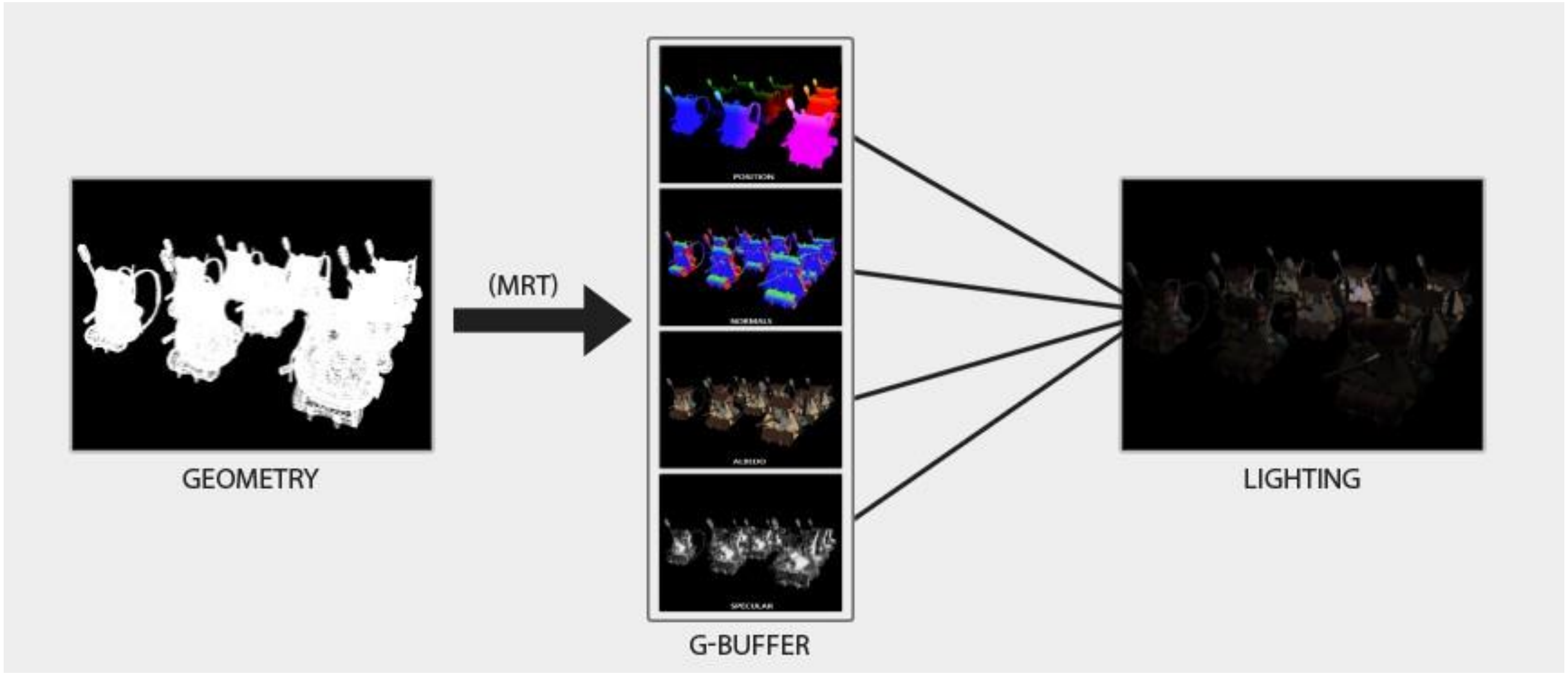
The Lighting Pass

Here’s the data we need to do simple shading:

- A 3D world-space **position** to calculate `light` and `view` vectors
- An RGB diffuse **color** vector also known as albedo.
- A 3D **normal** vector
- A **specular intensity** float.
- All light source position and color vectors.
- The player or viewer's position vector.

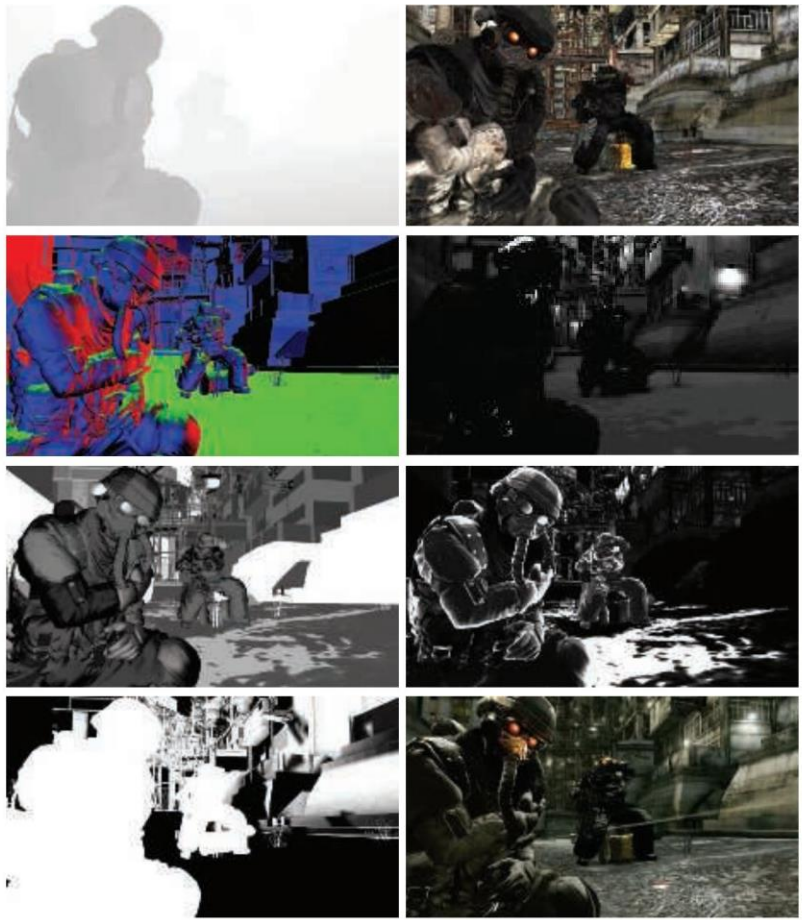


The Lighting Pass



MRT = Multiple Render Target

Another Example….



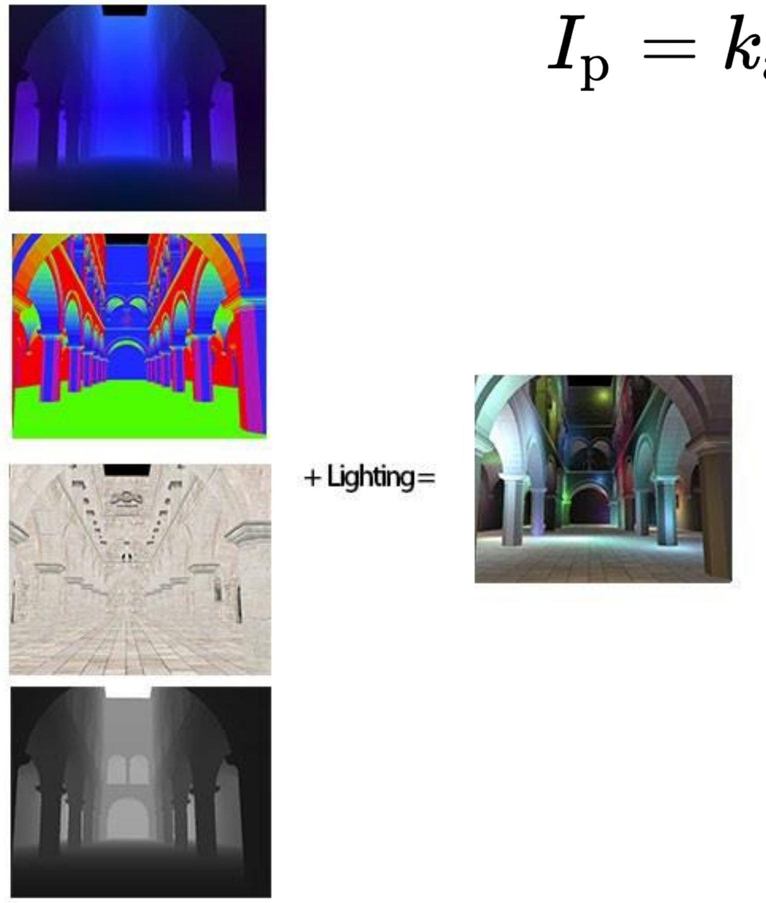
Left column, top to bottom: depth map, normal buffer, roughness buffer, and sunlight occlusion.

Right column: texture color (a.k.a. albedo texture), light intensity, specular intensity, and near-final image (without motion blur).

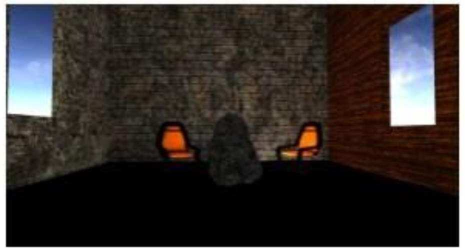
“*Killzone 2* (2009) was widely anticipated prior to its release. It was critically acclaimed by critics and fans, who praised it as a superior title to the original *Killzone*.”
- Wikipedia

Another Example…

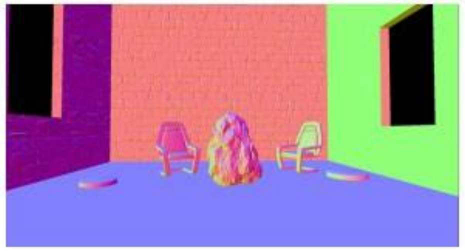
$$I_\mathrm{p} = k_\mathrm{a} i_\mathrm{a} + \sum_{m \,\in\, \mathrm{lights}} \left(k_\mathrm{d} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_\mathrm{s} (\hat{R}_m \cdot \hat{V})^\alpha i_{m,\mathrm{s}}\right)$$



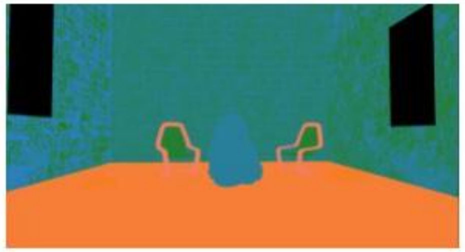
Actual UE4 G-Buffers



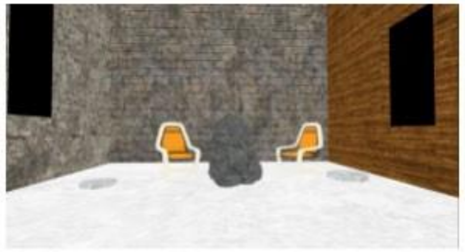
SceneColorDeferred: Contains Indirect lighting



GBufferA: World space normals, stored as RGB10A2_UNORM. Doesn't seem to use a particular encoding

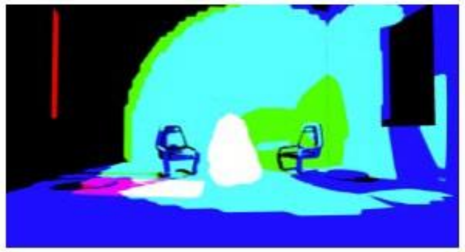


Distortion: Various material properties (metalness, roughness, specular intensity and shading model)



GBufferC: Albedo in RGB, AO in Alpha

GBufferE: Custom Data based on the shading model (eg



GBufferD: Pre baked shadowing factors

Fat Framebuffers

- The obvious render target layout goes something like:

– Position	A32B32G32R32F
– Normal	A16B16G16R16F
– Diffuse color	A8R8G8B8
– Material parameters	A8R8G8B8

- This adds up to 256 bits per pixel.
  At 1024x768, that is 24 MB, even without antialiasing!

- Also, current hardware doesn’t support mixing different bit depths for MRTs (possibly no longer true)

Framebuffer Size Optimizations

- Store normals in A2R10G10B10 format
- Material attributes could be palettized,
  then looked up in shader constants or a texture
- No need to store position as a vector3:
  - Each 2D screen pixel corresponds to a ray from the eyepoint into the 3D world (think raytracing)
  - You inherently know the 2D position of each screen pixel, and you know the camera setting
  - If you write out the distance along this ray, can reconstruct the original worldspace position

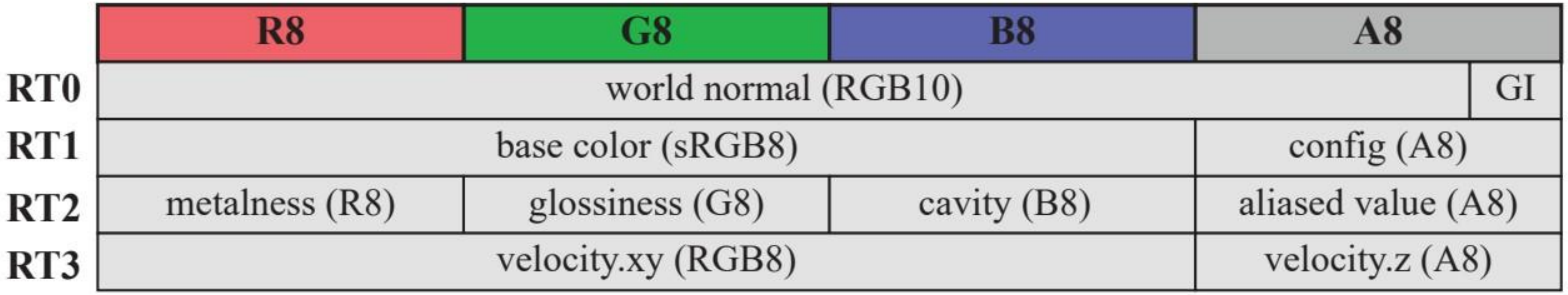
Shawn’s 2004 Framebuffer Choices

- 128 bits per pixel = 12 meg @ 1024x768:

– Depth	R32F
– Normal + scattering	A2R10G10B10
– Diffuse color + emissive	A8R8G8B8
– Other material parameters	A8R8G8B8

ILLINOIS

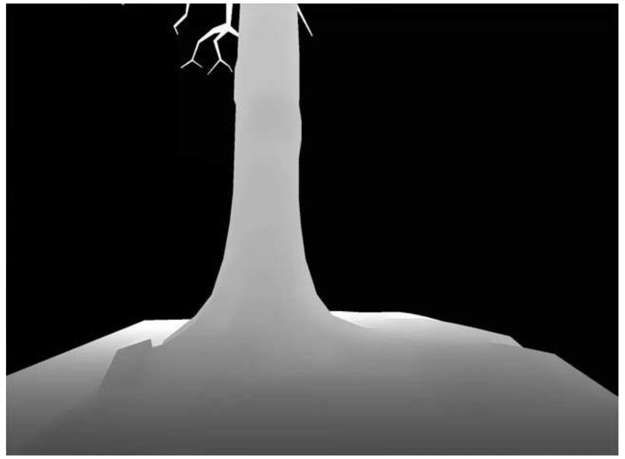
A More Modern Example



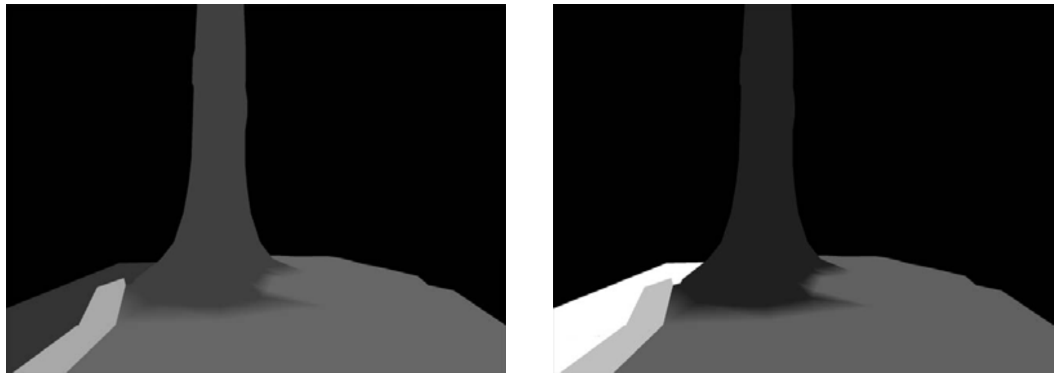
	R8	G8	B8	A8
RT0	world normal (RGB10)			GI
RT1	base color (sRGB8)			config (A8)
RT2	metalness (R8)	glossiness (G8)	cavity (B8)	aliased value (A8)
RT3	velocity.xy (RGB8)			velocity.z (A8)

An example of a possible G-buffer layout, used in Rainbow Six Siege (2015). In addition to depth and stencil buffers, four render targets (RTs) are used as well. As can be seen, anything can be put into these buffers. The “GI” field in RT0 is “GI normal bias (A2).”

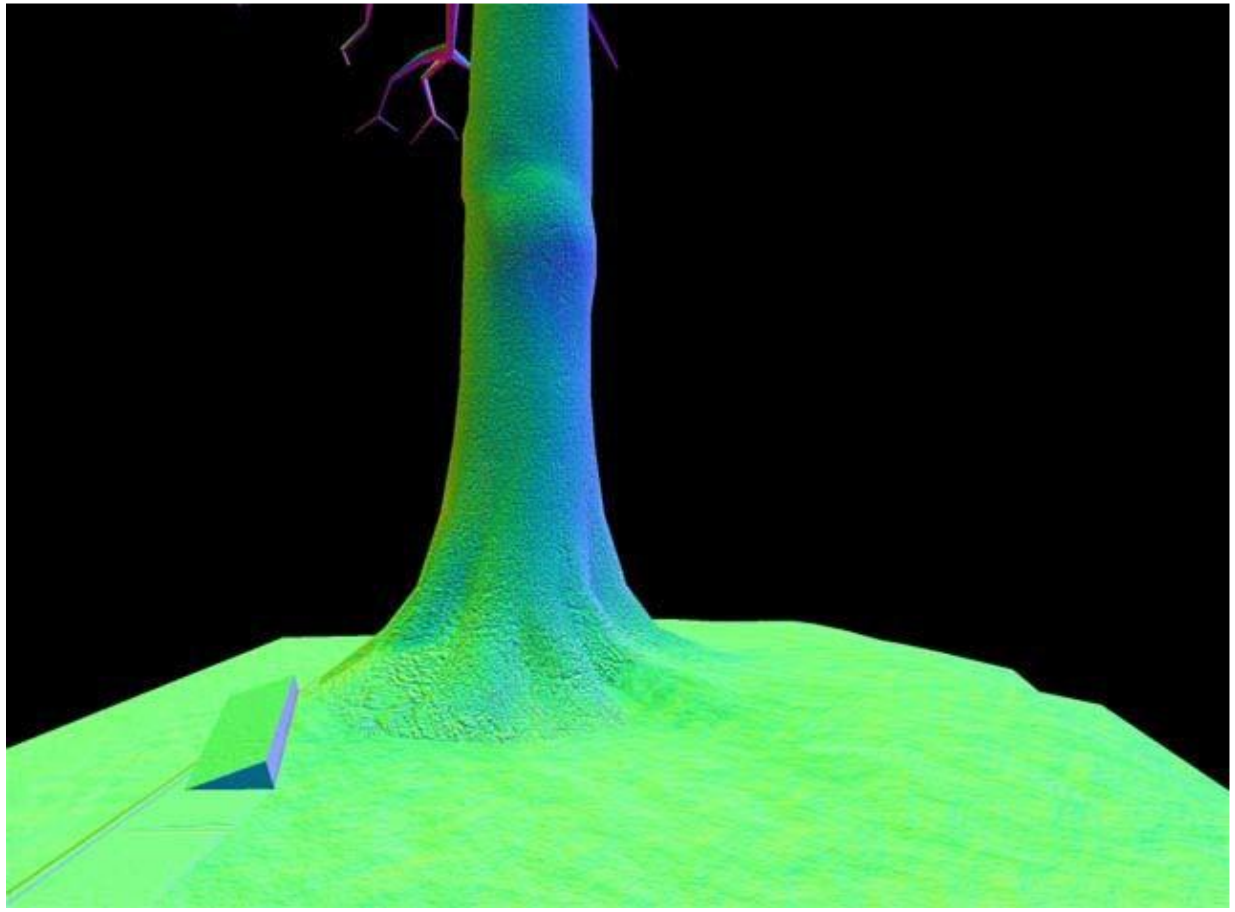
Depth
Buffer



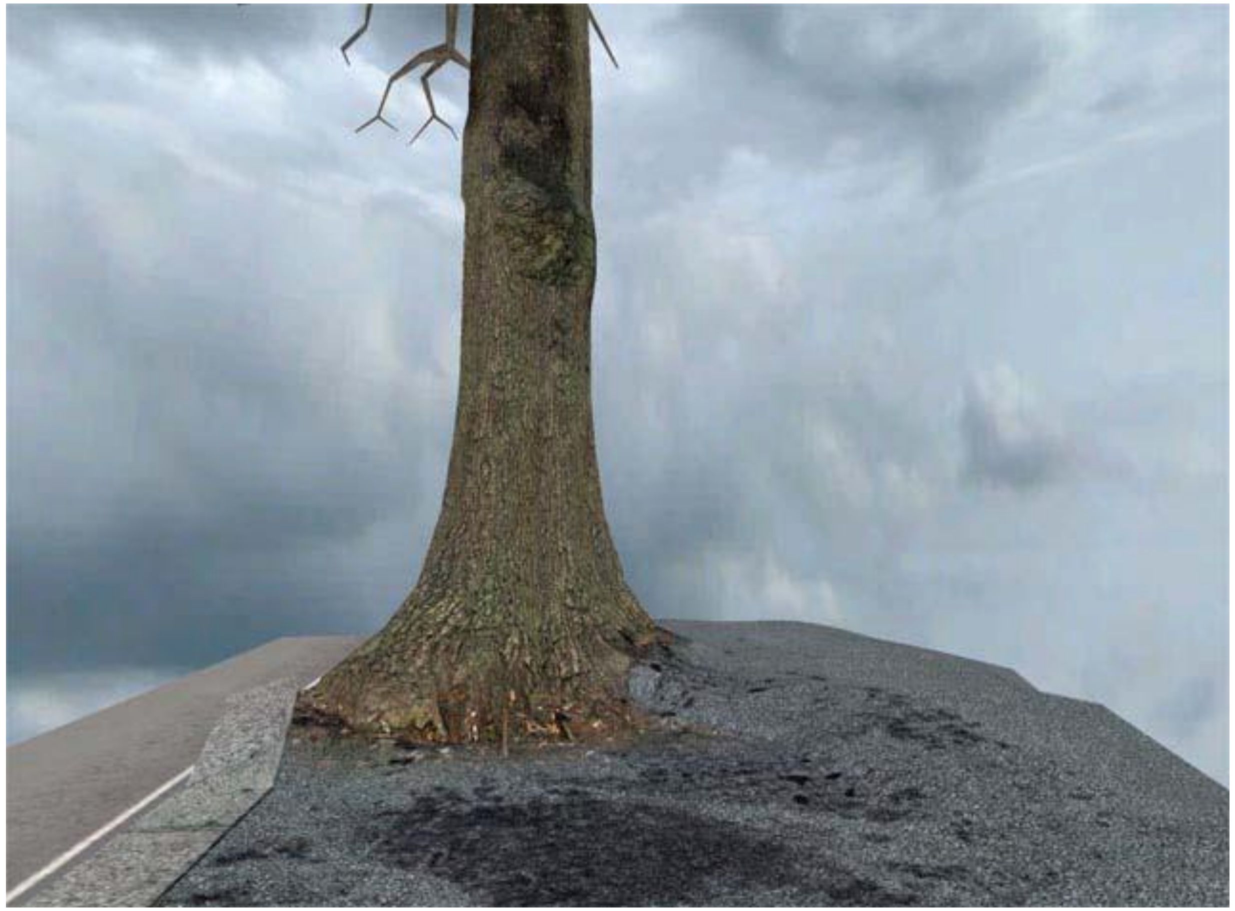
Specular Intensity and Power



Normal Buffer



Diffuse Color Buffer



Deferred Shading Results

Lighting Pass

One method is to apply each light one by one, using the Gbuffers to compute its effect.

- For each light we draw a screen-filling quadrilateral and access the G-buffers as textures.
- At each pixel, determine the location of the closest surface and if it is range of the light.
- If it is, we compute the effect of the light and place the result in an output buffer.
- We do this for each light in turn, adding its contribution via blending.
- At the end, we have all lights’ contributions applied.

Problems….

This process is about the most inefficient way to use G-buffers

Every stored pixel is accessed for every light, similar to how basic forward rendering applies all lights to all surface fragments.

Such an approach can end up being slower than forward shading, due to the extra cost of writing and reading the G-buffers

Optimization: Light Volumes

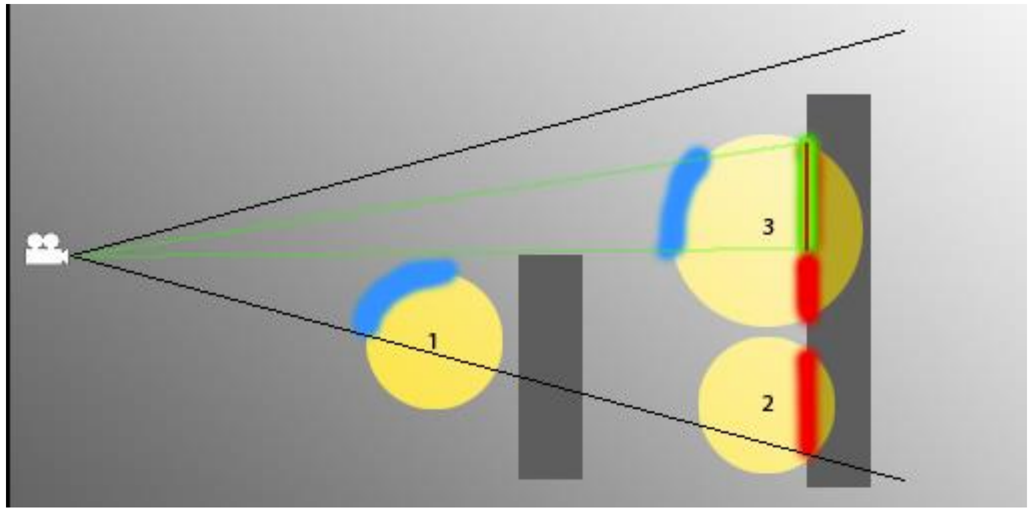
Determine the screen bounds of a light volume (a sphere)

Use them to draw a screen-space quadrilateral or rough sphere that covers those screen space bounds

Pixel processing is reduced, often significantly.

We can also use the third screen dimension, z-depth.

By drawing a rough sphere mesh encompassing the volume,
we can trim the sphere's area of effect further still



For example, if the sphere is hidden by the depth buffer,
the light's volume is behind the closest surface and so has no effect.

To generalize, if a sphere's minimum and maximum depths at a pixel do not overlap the closest surface,
the light cannot affect this pixel.

Global Lights

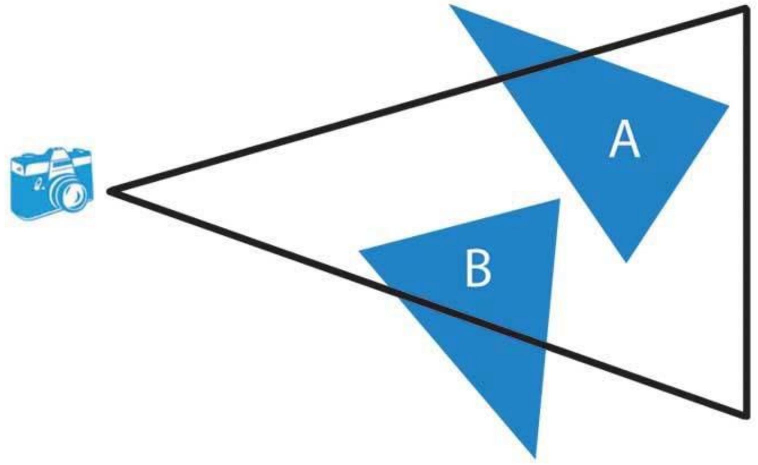
- Things like sunlight and fog affect the entire scene
- Draw them as a fullscreen quad
- Position, normal, color, and material settings are read from texture inputs
- Lighting calculation is evaluated in the pixel shader
- Output goes to an intermediate lighting buffer

Convex Light Hulls

The light bounding mesh will have two sides,

- but it is important that each pixel only gets lit once

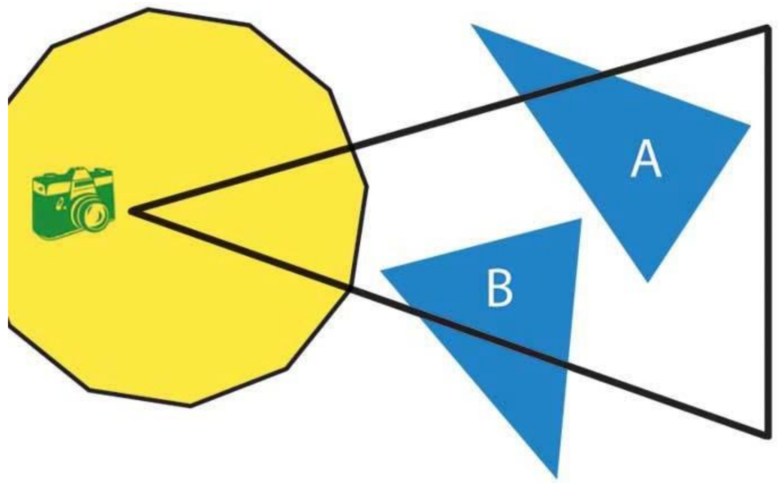
As long as the mesh is convex, backface culling can take care of this



Keep in mind…this code is running in the pixel shader…will only run if a polygon is covering the pixel…will run more than once if 2 polygons cover the pixel.

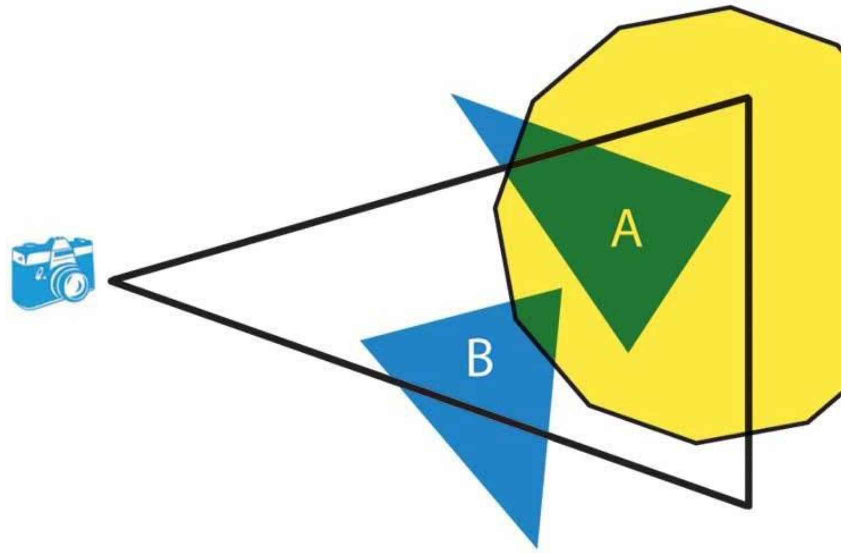
Convex Light Hulls #2

If the camera is inside the light volume, draw backfaces



Convex Light Hulls #3

- If the light volume intersects the far clip plane, draw frontfaces



Deferred Shading Advantages

- Excellent batching
- Invisible fragments not shaded
- Shade each visible fragment exactly once per light volume enclosing pixel
- Easy to add new types of lighting to shader
- Other kinds of post-processing (bloom?) are just special lights
  - Fit neatly into the existing framework

The Dark Side

- Alpha blending is a nightmare!
- Framebuffer bandwidth can easily get out of hand
- Can’t take advantage of hardware multisampling
- Forces a single lighting model across the entire scene (everything has to be 100% per-pixel)
- But will be increasingly attractive in the future…

This is from 2004…the future already arrived several years ago…

To Be More Precise….

- Two important limitations of deferred shading are transparency and antialiasing.
- Transparency is not supported in a basic deferred shading system
  - We can store only one surface per pixel.
- It is possible to now store lists of transparent surfaces for pixels and use a pure deferred approach
  - The norm is to mix deferred and forward shading as desired for transparency and other effects.
- An advantage of forward methods is that antialiasing schemes such as MSAA are easily supported.
- Deferred shading could store all N samples per element in the G- buffers to perform antialiasing
  - But this increases in memory cost, fill rate, and computation make this approach expensive.