COMP3421

Postprocessing, Splines

Robert Clifton-Everest

Email: robertce@cse.unsw.edu.au

Postprocessing

- Adding visual affects (lighting, colouring, stylisation…) after the scene has already been rendered.

- Uses techniques from image processing to achieve this.

- Modern games rely on this heavily (~50% of the frame budget)

Frame Buffer Objects

- The framebuffer OpenGL gives us by default is very limited in most implementations.

- It has:
  - No alpha channel
  - Only 8 bits per colour channel
  - Limited precision depth buffer

Frame Buffer Objects

- We can create our own frame buffer objects with:

```
int[] fbos = new int[1];
gl.glGenFramebuffers(1, fbos, 0);
```

- In order to use the frame buffer we have to attach to it at least a color buffer and optionally a depth or other buffers.

Frame Buffer Objects

- A color buffer can just be a regular texture

```
gl.glTexImage2D(GL.GL_TEXTURE_2D, 0,
GL.GL_RGBA, width, height, 0, GL.GL_RGBA,
GL.GL_UNSIGNED_BYTE, null);
```

- …which we attach to the frame buffer

gl.glFramebufferTexture2D(GL.GL_FRAMEBUFFER, **GL.GL_COLOR_ATTACHMENT0**, GL.GL_TEXTURE_2D, texID, 0);

Render Buffer

- A render buffer is like a texture, but there's no way to read from it within a shader. OpenGL is free to store it in whatever format works best for that implementation.

  - Suppose we wanted a render buffer that gave 24 bits for depth

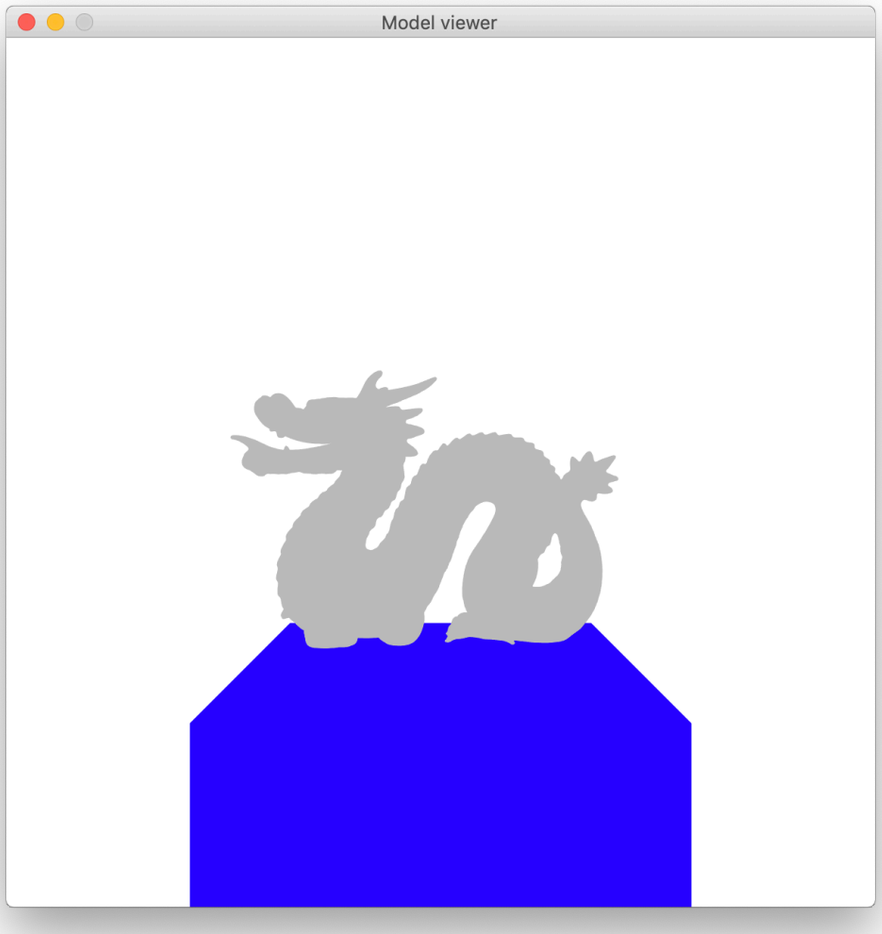
```
int[] rbos = new int[1];
gl.glGenRenderbuffers(1, rbos, 0);
gl.glBindRenderbuffer(GL.GL_RENDERBUFFER, rbos[0]);
gl.glRenderbufferStorage(GL.GL_RENDERBUFFER,
GL.GL_DEPTH_COMPONENT24, width, height);
```

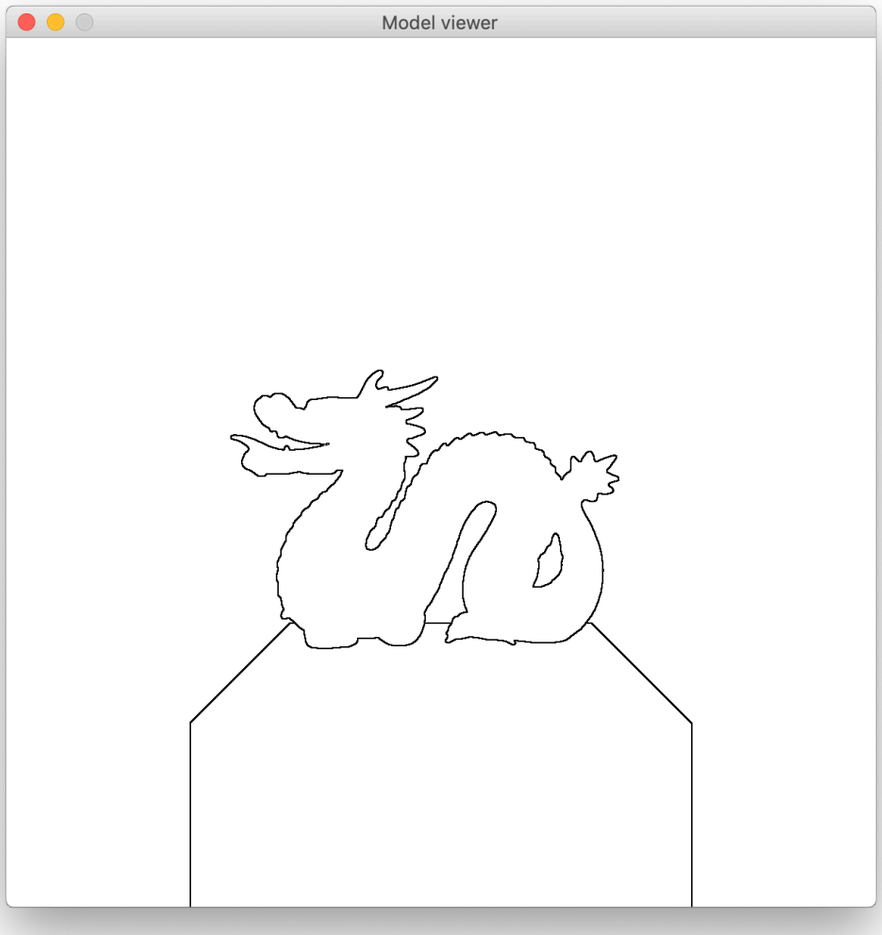
Render Buffer

- It still needs to be attached to the FBO

```
gl.glFramebufferRenderbuffer(GL.GL_FRAMEBUFFER,
GL3.GL_DEPTH_ATTACHMENT, GL.GL_RENDERBUFFER,
rbos[0]);
```

Simple Edge Detection

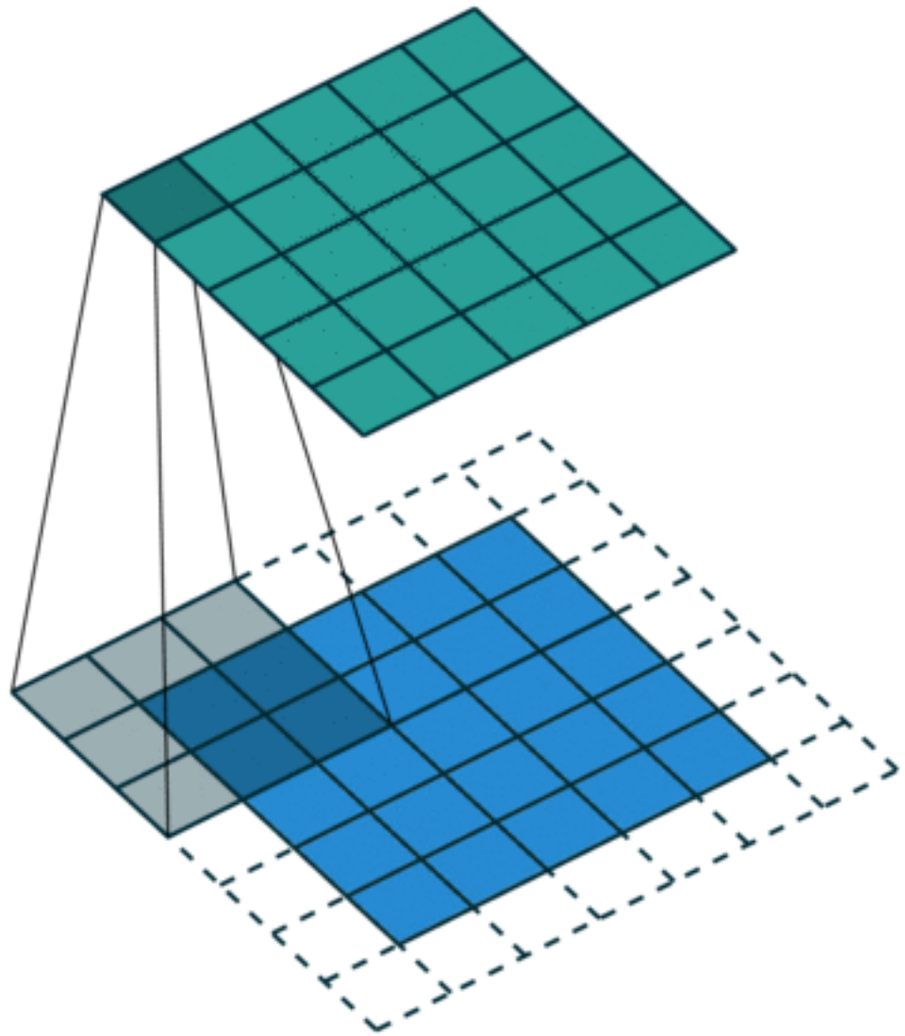




Stencil operations

Note: This has nothing to do with the stencil buffer

- Computing new values based on surrounding values.
- Useful technique in many areas
- What about the edges?
  - Just use GL_CLAMP_TO_EDGE



Sobel operator

- Multiply each element in the 3x3 neighbourhood with this kernel to get the derivative in the x-direction

-1	0	1
-2	0	2
-1	0	1

- Then this kernel to get the derivative in the y-direction

-1	-2	-1
0	0	0
1	2	1

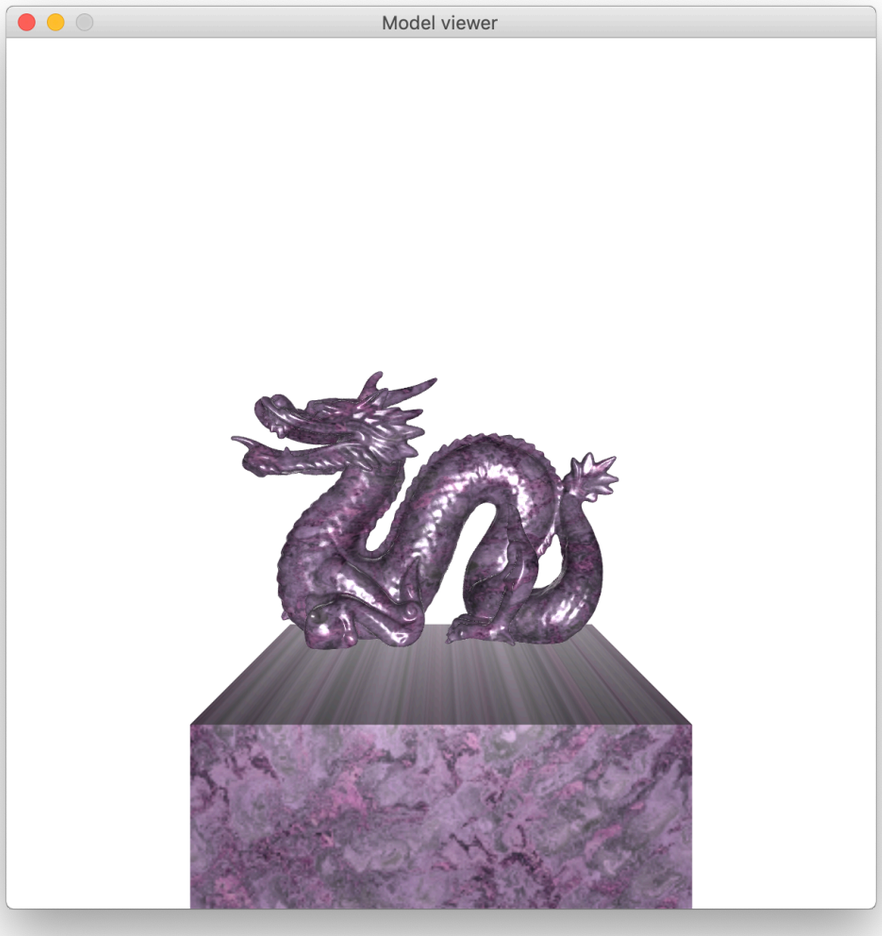
- Using the derivates, calculate the total magnitude for each color component

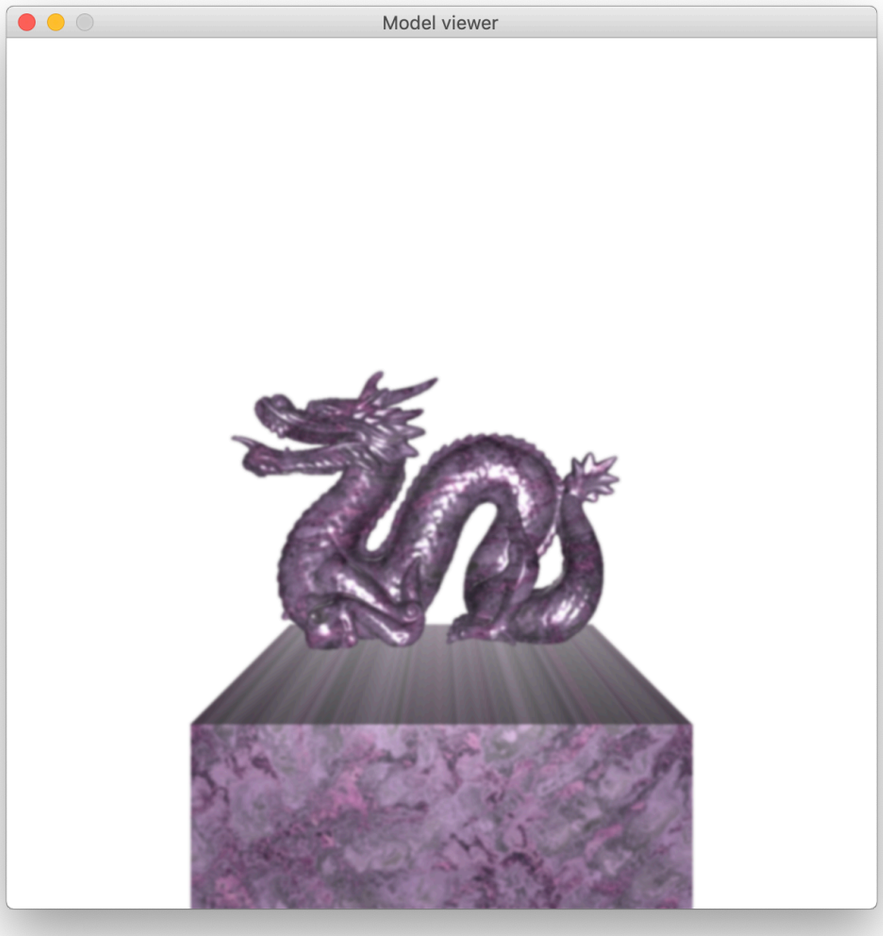
Sobel operator

- If the magnitude for a color component is greater than some threshold, treat it as an edge.

- See ModelViewer.java and fragment_sobel.glsl

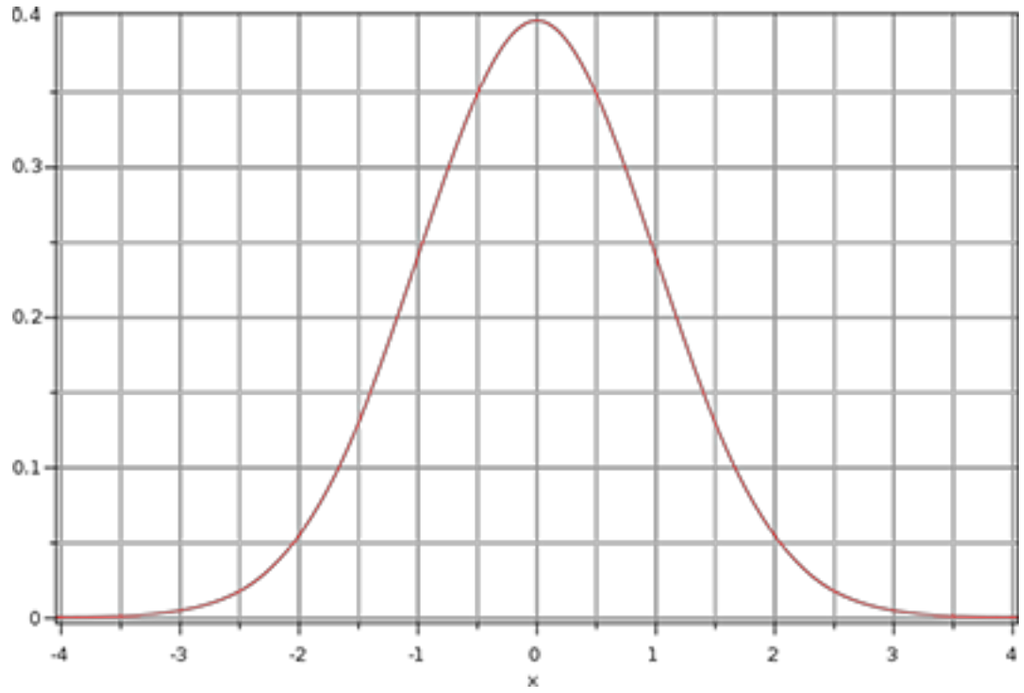
Gaussian blur





Gaussian blur

- Uses a Gaussian distribution to combine each pixel with its neighbours



- The pixel itself is weighted most, and nearby pixels weighted depending on their distance from it.

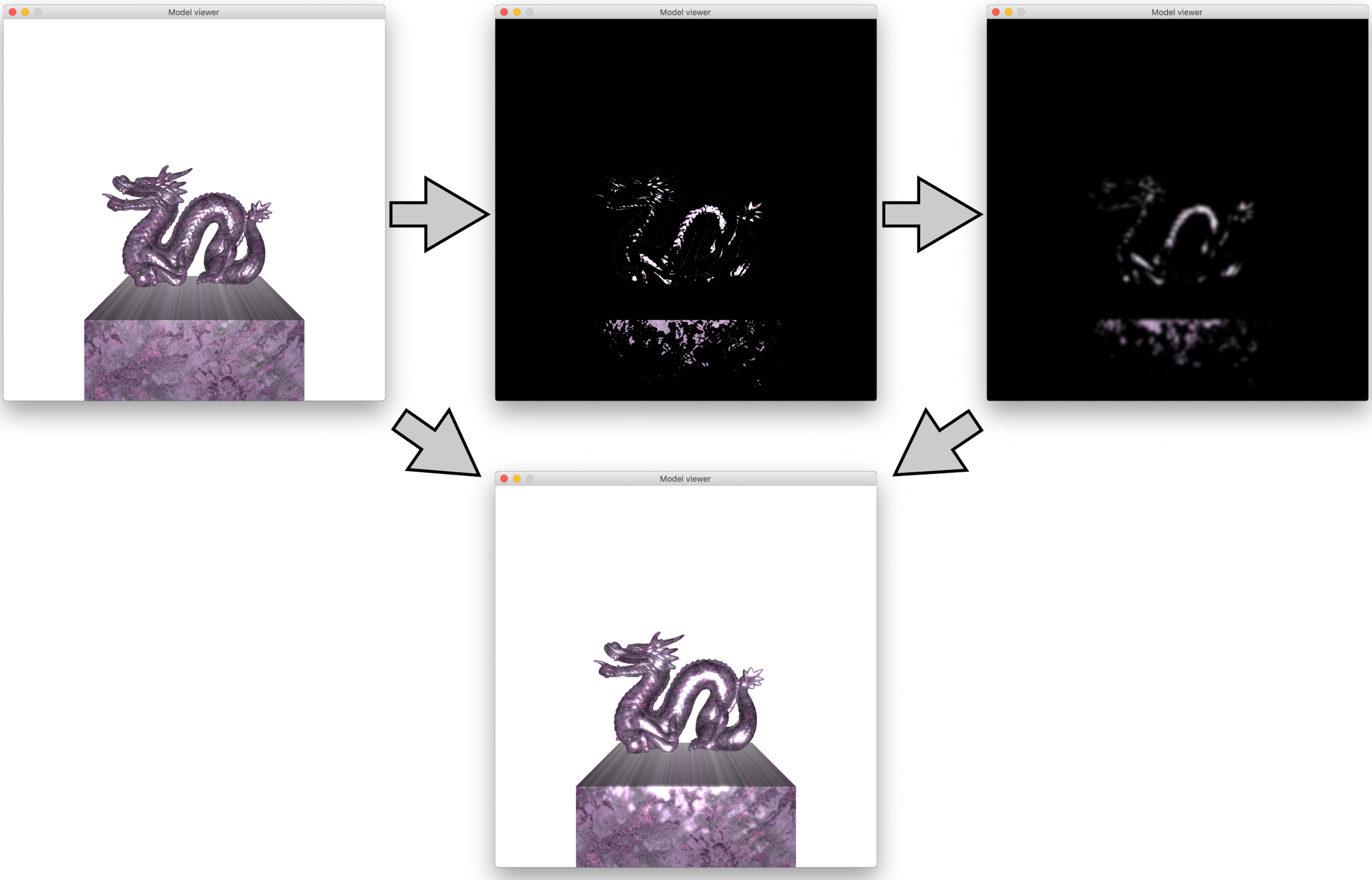
Gaussian blur

- Best done in 2 passes.
  - Blur in the x-direction (9x1 kernel)
  - Then blur in the y-direction (1x9 kernel)
- Further blurring can be achieved by using a larger kernel or by repeating the process
- See fragment_blur.glsl

Bloom filter

- Combines multiple post-processing steps.
- Step 1. Extract bright regions.
- Step 2. Blur bright regions.
- Step 3. Draw original scene to screen
- Step 4. Draw blurred brightness on top

Bloom filter



High Dynamic Range (HDR)

- By using FBOs, we can render into buffers with more color detail (e.g. 16 bits per color channel instead of 8)

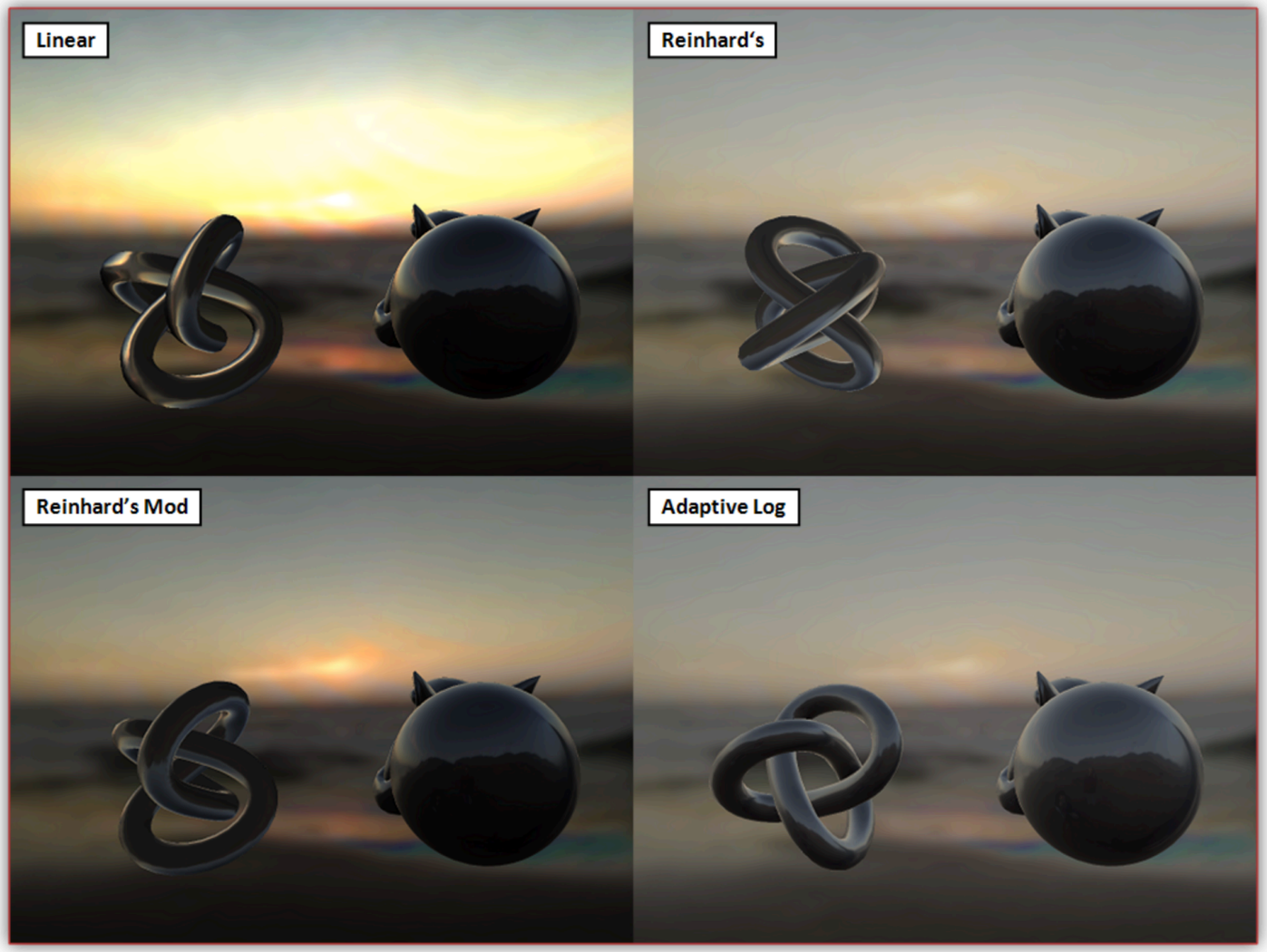
```
gl.glTexImage2D(GL.GL_TEXTURE_2D, 0,
GL.GL_RGBA16F, width, height, 0, GL.GL_RGBA,
GL.GL_UNSIGNED_BYTE, null);
```

- To render to the screen, this colour information has to be mapped to the conventional 8 bits per channel.

- This process is referred to as tone mapping

High Dynamic Range (HDR)

- Different tone mappings can do things like reduce overall brightness or enhance certain colours



High Dynamic Range (HDR)

- See here for more details: https://learnopengl.com/Advanced-Lighting/HDR

Ambient Occlusion

- Approximates indirect lighting by identifying small holes, crevices and gaps (i.e. places light doesn't get into) and making them darker.
- Most common form is Screen-Space Ambient Occlusion (SSAO). First popularised by Crysis (2007).

Ambient Occlusion

- Relies upon being able to read from the depth buffer after a scene has been rendered.

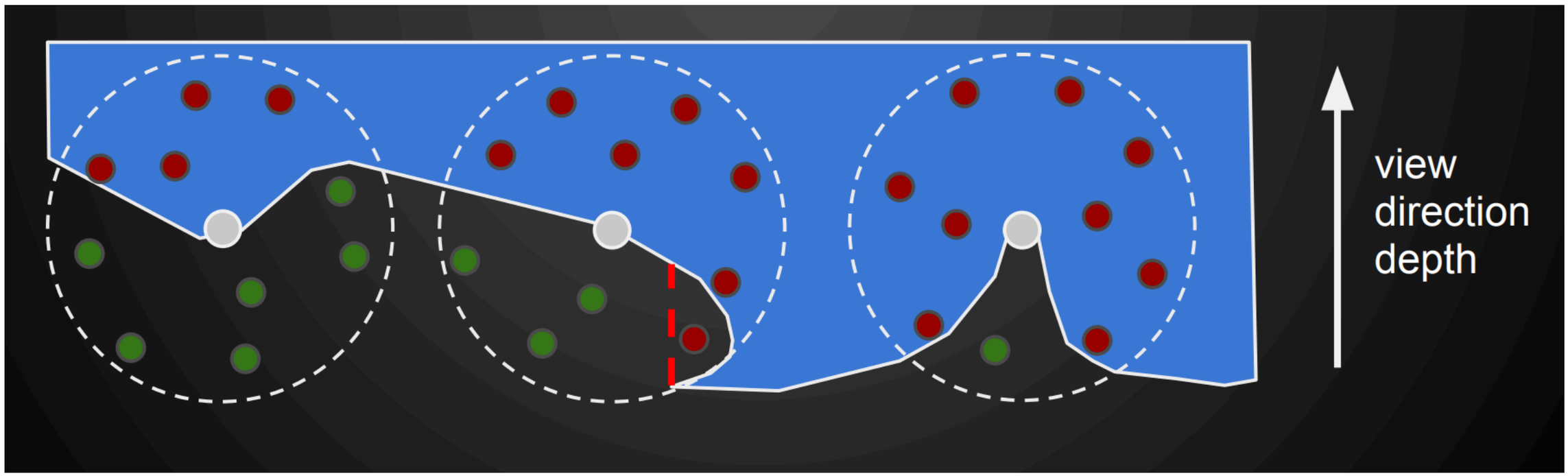
- For each pixel that has already been rendered, calculate an occlusion factor based on the depth of it and the depth of surrounding pixels.

- Which surrounding pixels?

  - Looks best if picked randomly from a sphere

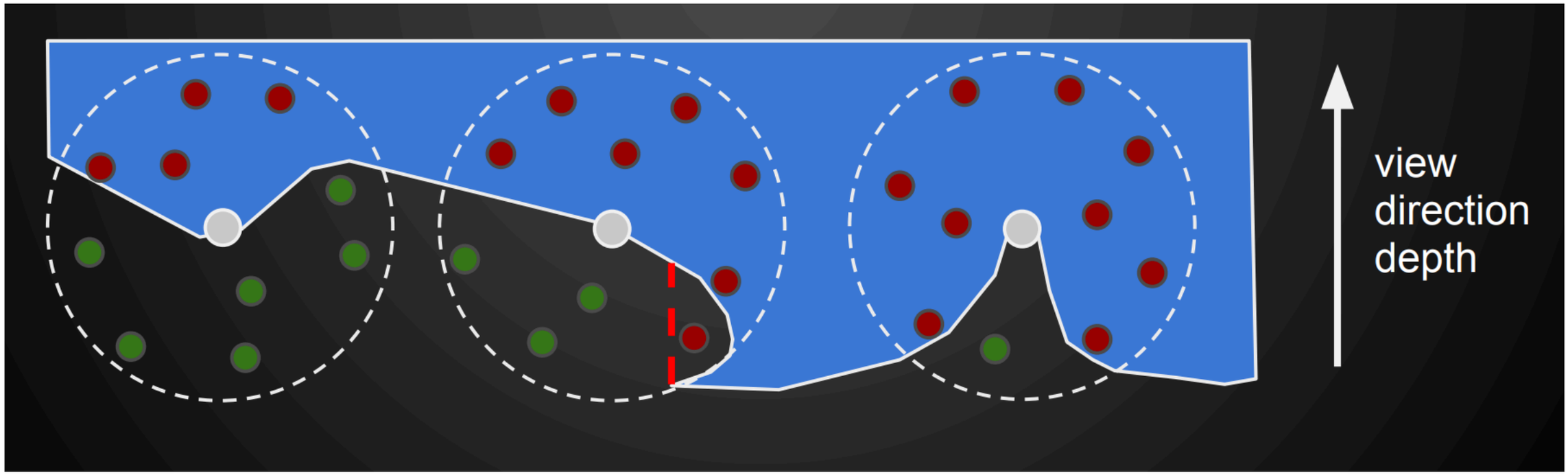
Ambient Occlusion

- If more than half of the random samples taken from a sphere are blocked by a value in the depth buffer then the pixel will be darkened.

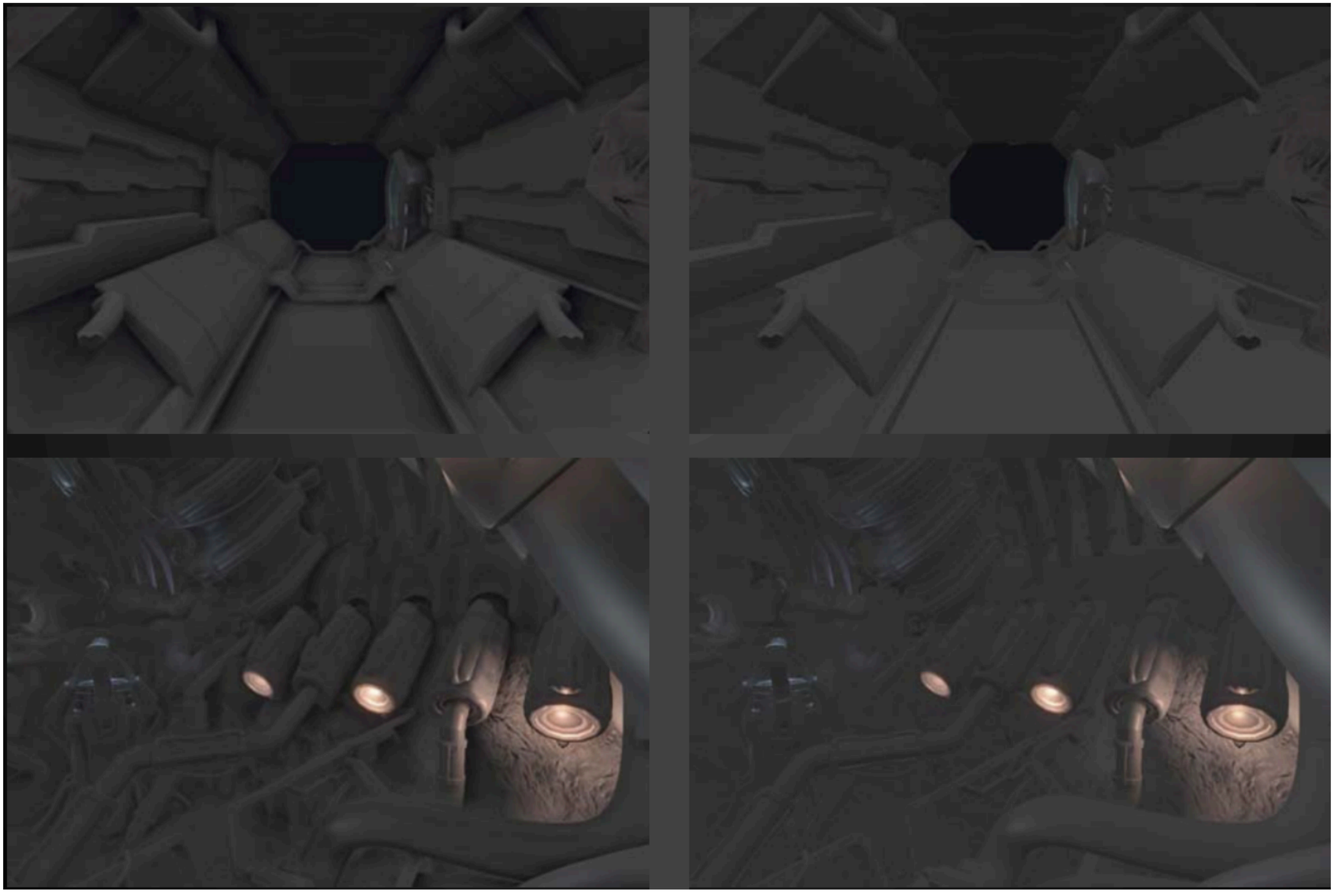


Ambient Occlusion

- Is inaccurate as it assumes geometry doesn't have gaps in the Z-direction.



Ambient Occlusion



Ambient Occlusion

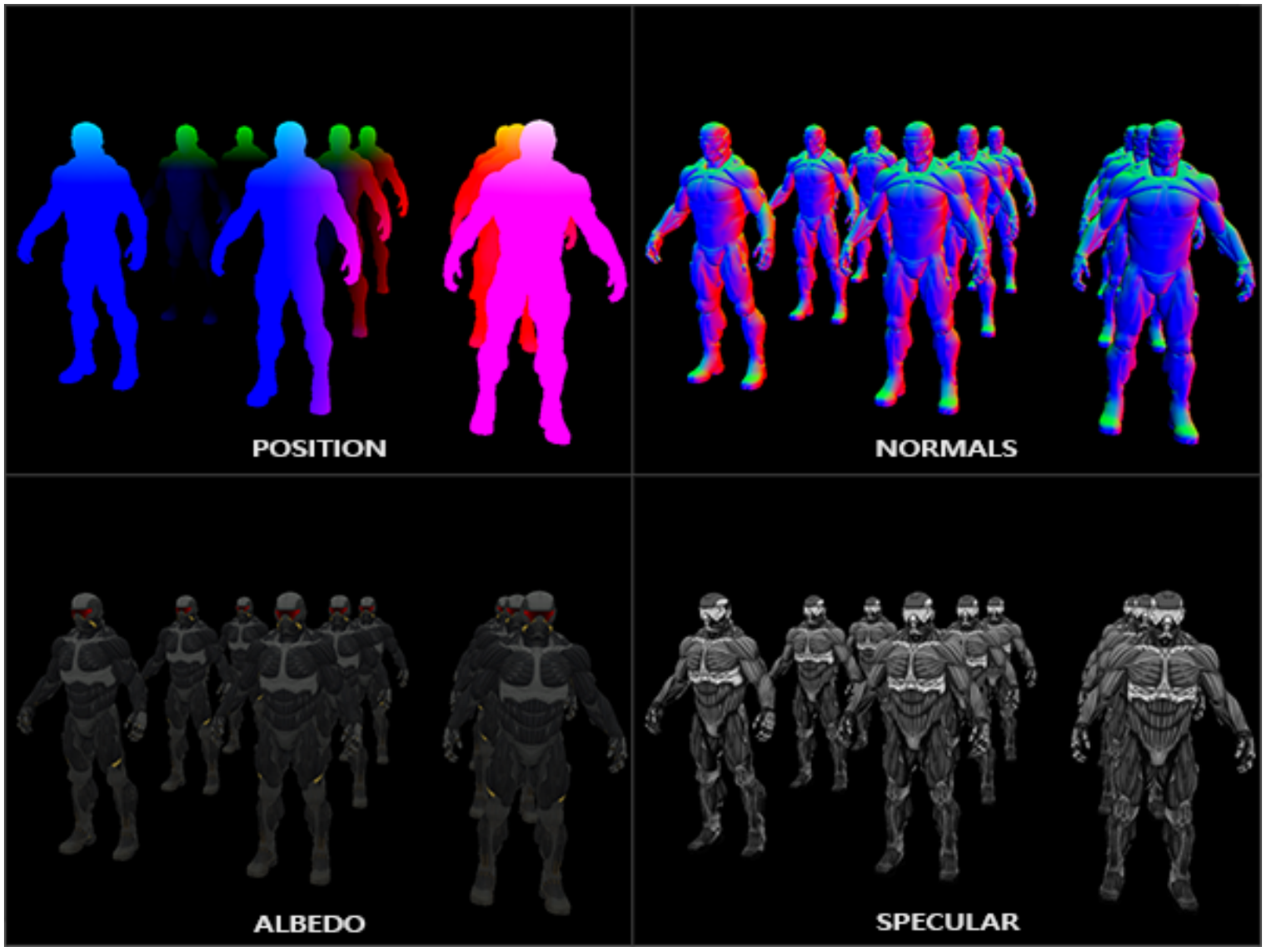
- More info: https://learnopengl.com/Advanced-Lighting/SSAO

Deferred Shading

- It's common with postprocessing to need information about a pixel other than its colour or its depth.
- In deferred shading we render the scene into multiple buffers containing basic information. E.g. position, normal, specular, depth

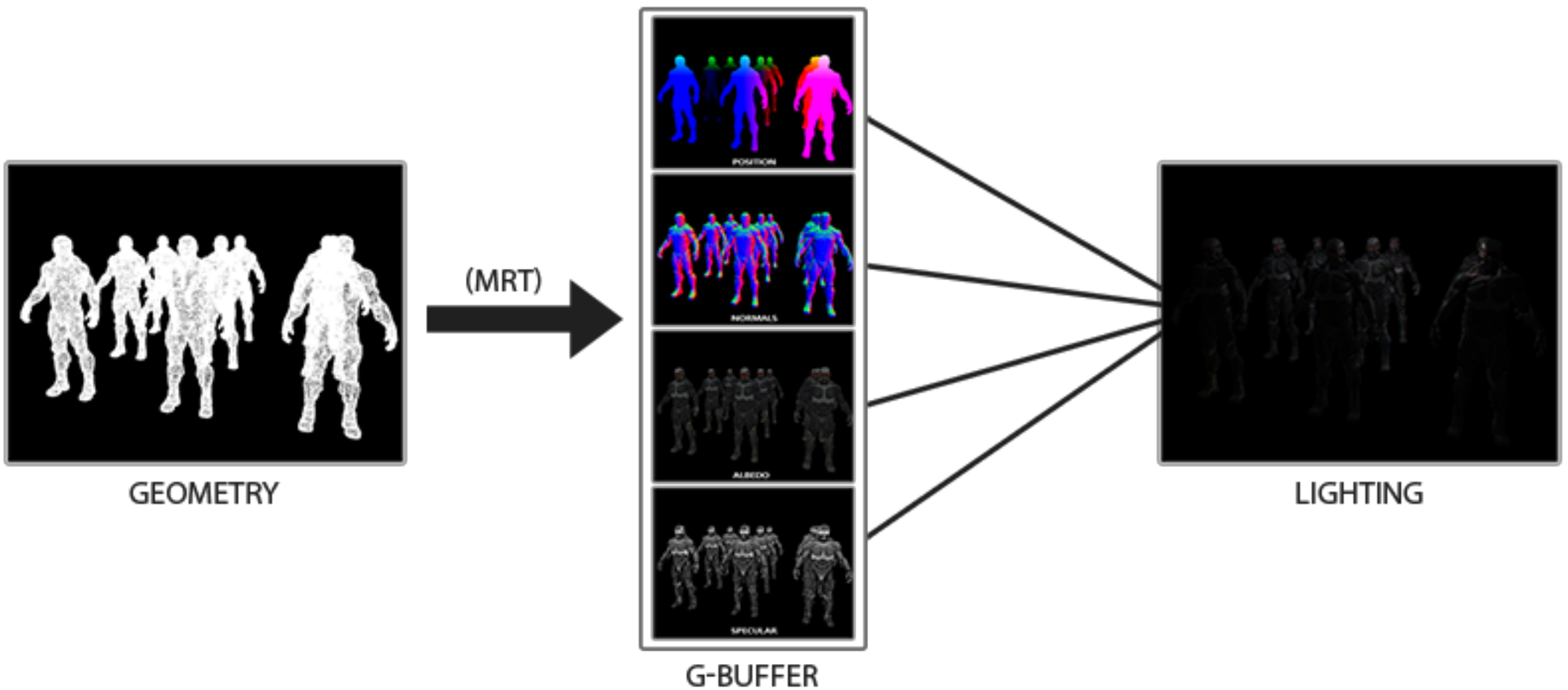
Deferred Shading

- The G-Buffer refers to all the different buffers generated.



Deferred Shading

- Actual lighting calculations can be deferred to a post processing step



Deferred Shading

- More info: https://learnopengl.com/Advanced-Lighting/Deferred-Shading