Prelighting

Mark Lee

what we did in the past

- standard multi-pass approach to lighting used on R1 and RCF
- lighting means dynamic lighting

```
for each light
{
        set light state
        for all meshes light intersects
        {
                render mesh
        }
}
```

multi-pass issues

- very slow, for each new light we are potentially re-rendering the entire scene
- much redundant work done such as repeated texture look ups
- each object we render needs to track lights which illuminate it
- hard to optimize, the RSX isn't great for vertex processing
- lessons learnt from the end of R1

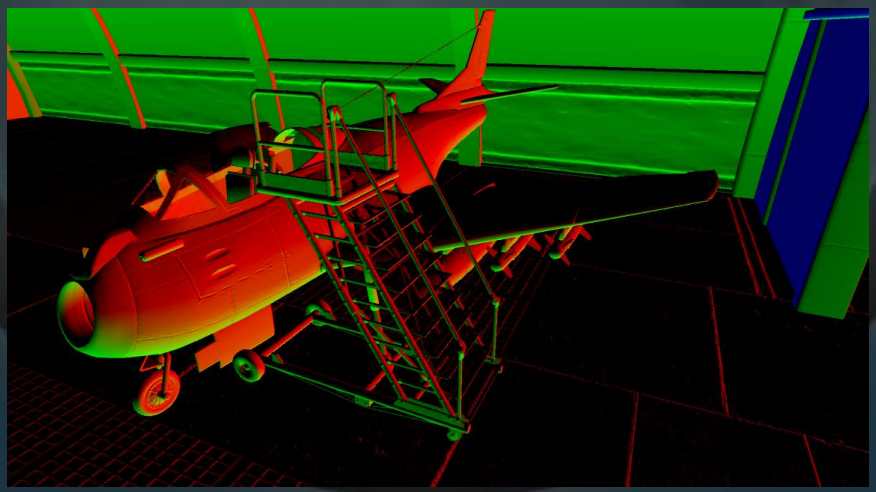
multi-pass issues

- what are we doing for each light?
- **diffuse = (n dot l) * albedo * x**
  - l is the normalized light direction
  - x is not important here but takes care of light properties, attenuation and shadows
- n is a function of:
  - normal map
  - parallax map
  - detail map
  - detail map mask
- albedo is a function of:
  - base map
  - parallax map
  - detail map
  - detail map mask
  - incandescence map
- lots of duplicated work!

idea – cache once and reuse

- create a frame buffer which stores the normal per pixel
- create another for the albedo per pixel
- only need to store values for the closest surface
- also need to store baked light, gloss, spec power plus other per material properties
- we can derive a 3D position for each pixel by using the 2D screen xy coordinates and a reconstructed linear depth
- rendering lights becomes a 2D post processing task
- added bonus – no need to track which lights affect which geometry anymore

inputs (screen captures)



result

deferred light problems

- the memory footprint is prohibitive, a 1280*720 MSAA buffer is 7.3mb, multiplied by 5 is 38mb
- unproven technology on the PS3
- a pretty drastic change to implement

prelighting

- turn it around so we leverage our current rendering pipeline
- the idea is to compute all the light contributions and cache to 2 separate buffers, diffuse and specular
- regular rendering passes can use these in place of the old multi-pass lighting
- only need to allocate memory for lighting buffers

prelighting stages

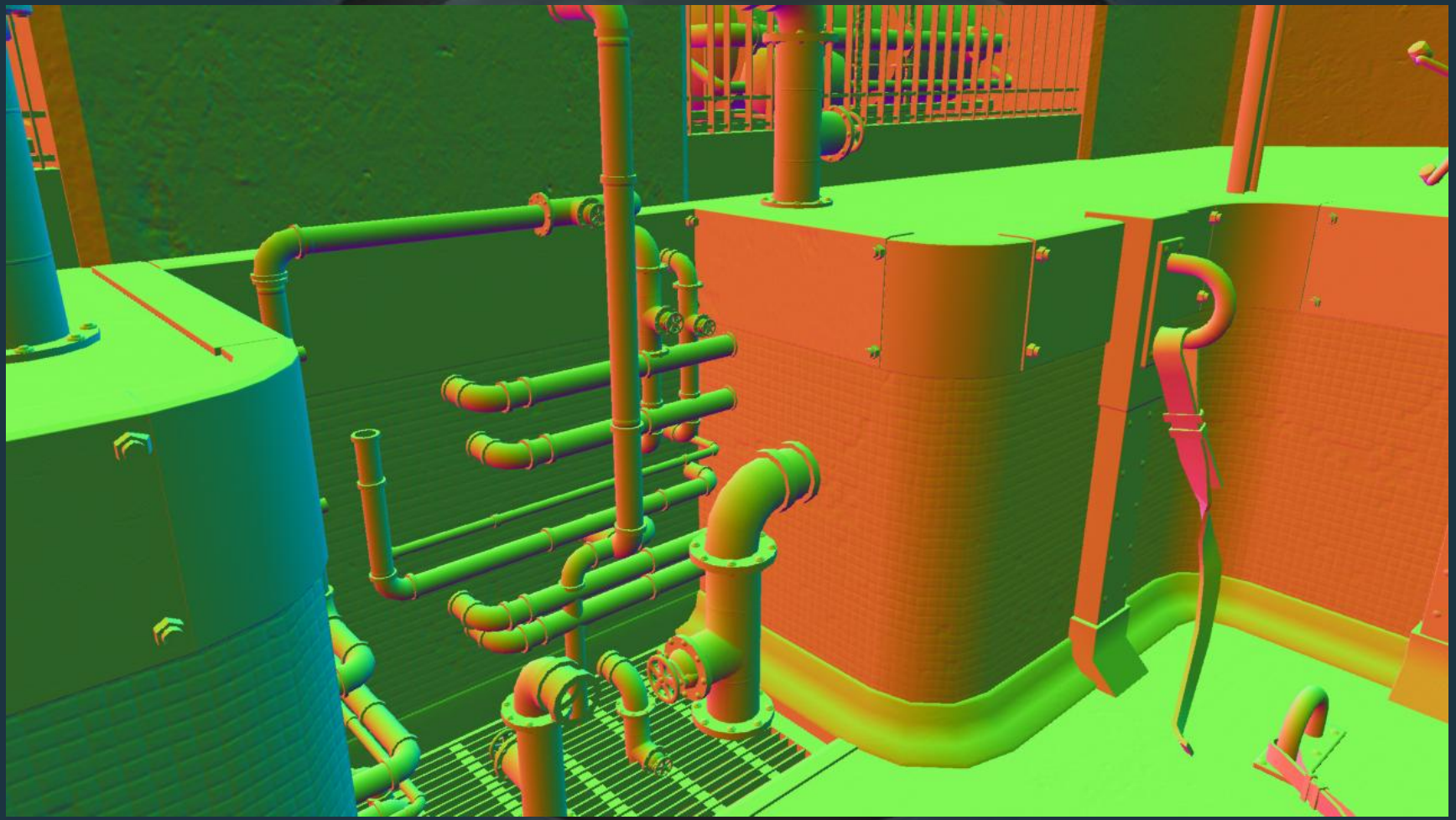
- **step (a) –** dnrm pass - render depth and normals for scene
- step (b) - render use these to render diffuse and specular light buffers
- step (c) - render the rest of the frame as before

step (a) – depth normal pass

- writes into our main render frame buffer and depth buffer – nothing is in there yet
- rgb = viewspace normal, can be complex to compute due to detail normal maps, detail map masks and parallax
- alpha = per material specular power
- we are only interested in the closest surface which is exactly what the depth buffer gives us

(a) depth

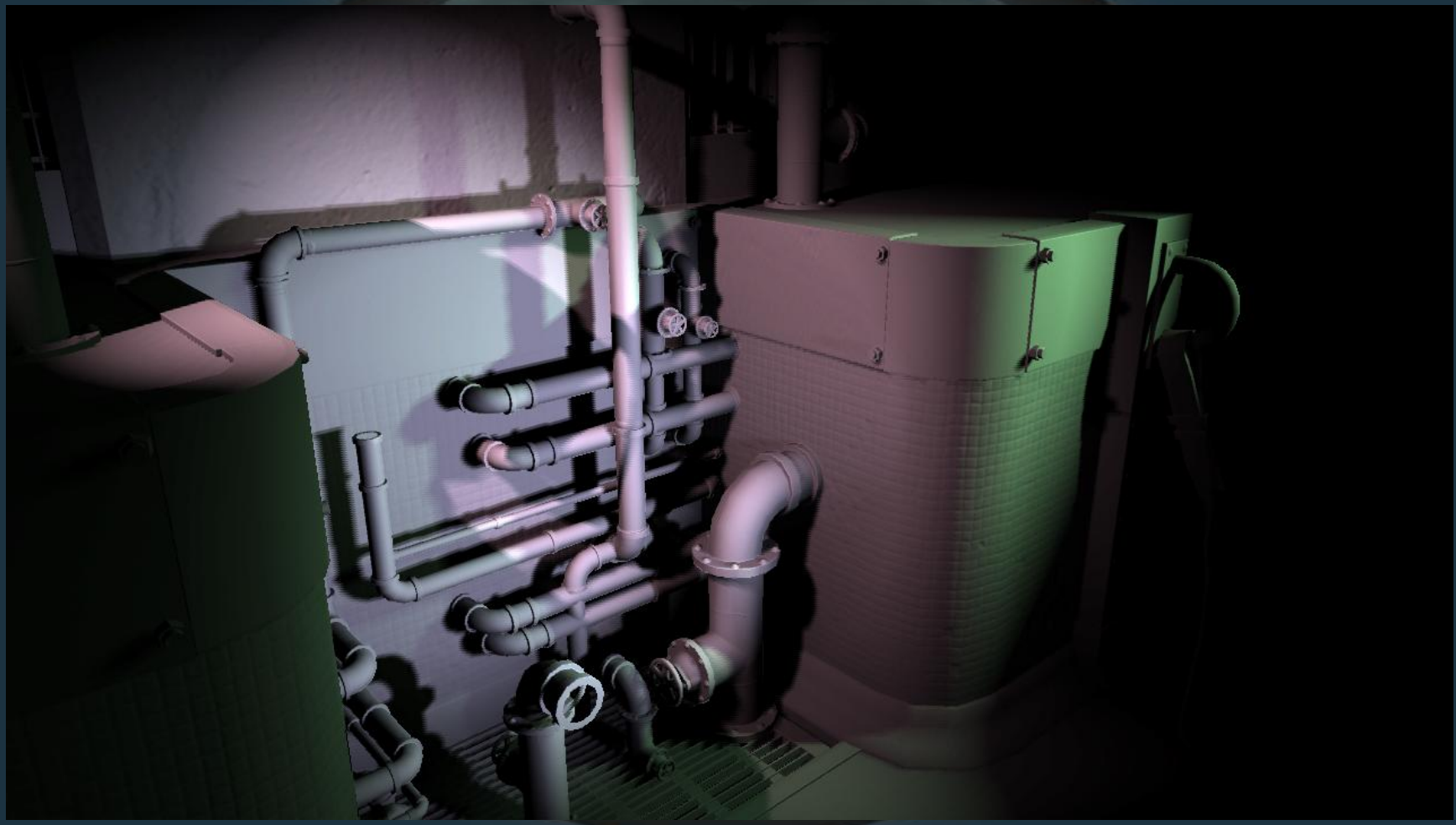
(a) normals



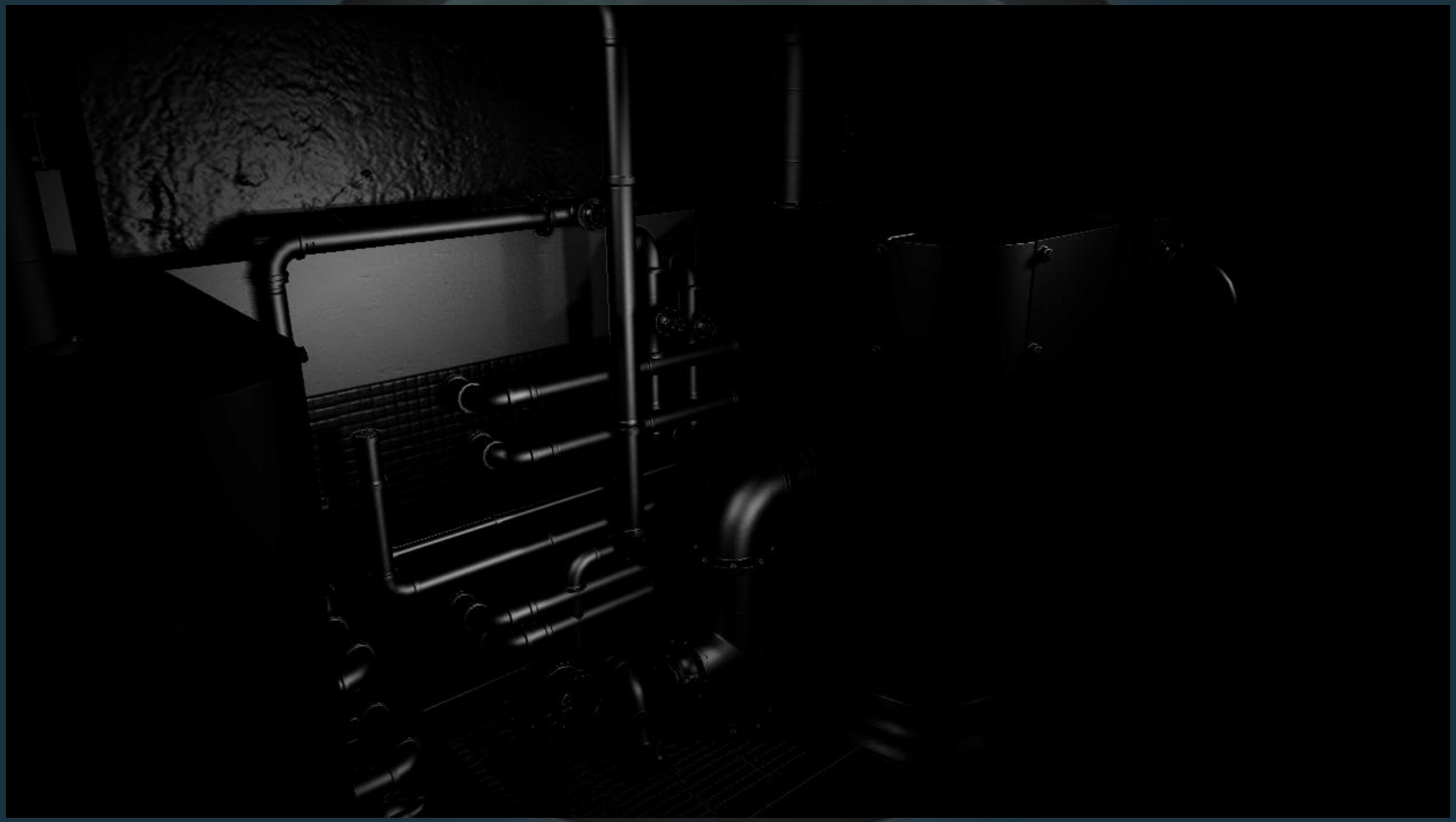
(b) prelighting pass

- this is where we perform the actual lighting
- point and spot lights supported
- one screen space projected quad drawn per light
- shadow maps are accumulated into these buffers, this allows us to free us the shadow memory earlier
- projected textures are also rendered into diffuse light buffer
- MRT used to write to diffuse and specular buffers simultaneously
- available to custom shaders if they need lighting also

(b) diffuse light



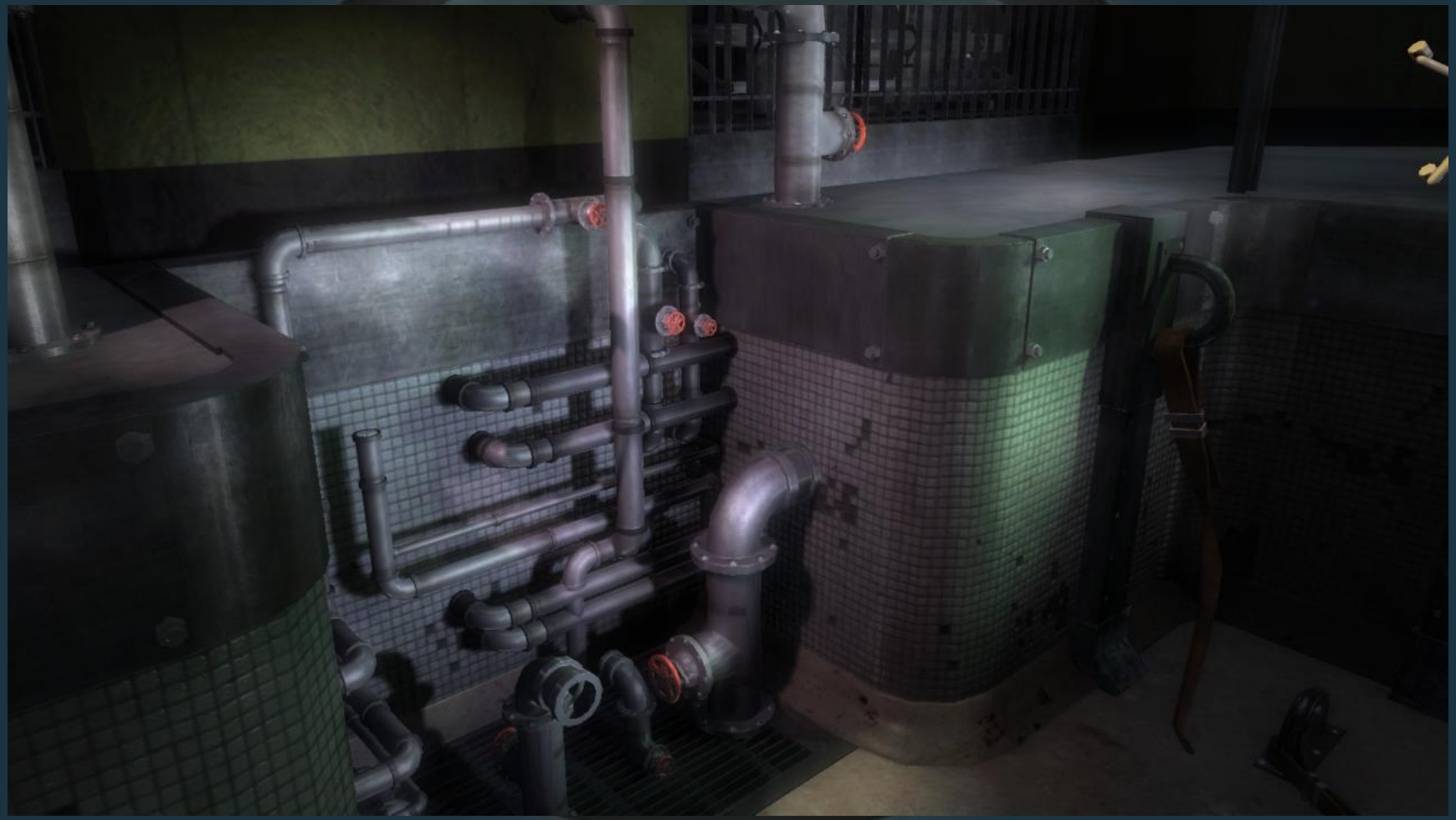
(b) specular light



(c) regular geometry pass

- remains identical to before except they look up the lighting buffers
- diffuse light is modulated with base map here
- specular is modulated by gloss map
- material specific specular properties as defined by the shader, for now it’s just a simple specular tint

(c) regular pass – composited light buffers



optimization – don’t light pixels which don’t need to be lit

optimization – don’t light pixels which don’t need to be lit

optimization – don’t light pixels which don’t need to be lit

- typically used to do stencil shadowing
- render light volume as double sided geometry

```
clear stencil buffer
if (front facing and depth test passes)
        increment stencil
if (back facing and depth test passes)
        decrement stencil
render light only to pixels which have non-zero stencil
```

- make sure light volumes are closed

optimization – don’t light pixels which don’t need to be lit

- same problems as stencil shadows have (and same solutions)
- problematic if we go inside the light volume
- use depth fail stencil test a.k.a. Carmack’s reverse
- only do this when we have to, we can’t use z-cull optimizations when doing this (if we did stencil updates could potentially get culled also)

optimization – don’t use MSAA light buffers

- primarily a memory optimization, saves ~7.4mb
- we still have to do lighting computations per sub-sample
- shadowing and projected texture computations only need to be done once per pixel
- requires a depth resolve

optimization – don’t use allocate a separate specular buffer

- instead use the alpha channel of the diffuse light buffer
- saves an additional 3.7mb, we allocate a single non-MSAA buffer instead of 2 MSAA buffers
- we lose per light specular color but still get material specific specular response
- the jury's still out on this one

optimization – don’t use allocate a separate specular buffer

- reference image

- monochrome specular

optimization – conditional rendering

- re-order depth pass into occluders and occludees
- when rendering the depth pass, count the number of pixels rendered for each fragment
- when we render our main pass, skip that fragment if associated pixel count is zero
- skipping the fragment is done near the top of the pipe – no vertex processing is executed
- fully done on the GPU, no PPU interaction necessary
- ~20% speed increase

optimization – trim the index buffers

- the depth pass is more of an immediate concern that the time taken to render lights
- >50% of vertices get culled after they already been transformed
- don’t waste time transforming those
- we could potentially save about ~30% on the depth pass
- memory requirements are problematic


questions?