Computer Graphics

COMP3421/9415
2021 Term 3 Lecture 17

What did we learn last week?

Reflections

- Reflective objects with:
- Environment Maps
- Cube Maps
- Realtime Cube Maps

Post Processing

- Framebuffers and Render Targets
- Kernel Sampling
- Bloom

What are we covering today?

Shadow Mapping

- Casting shadows in Polygon Rendering

Deferred Rendering

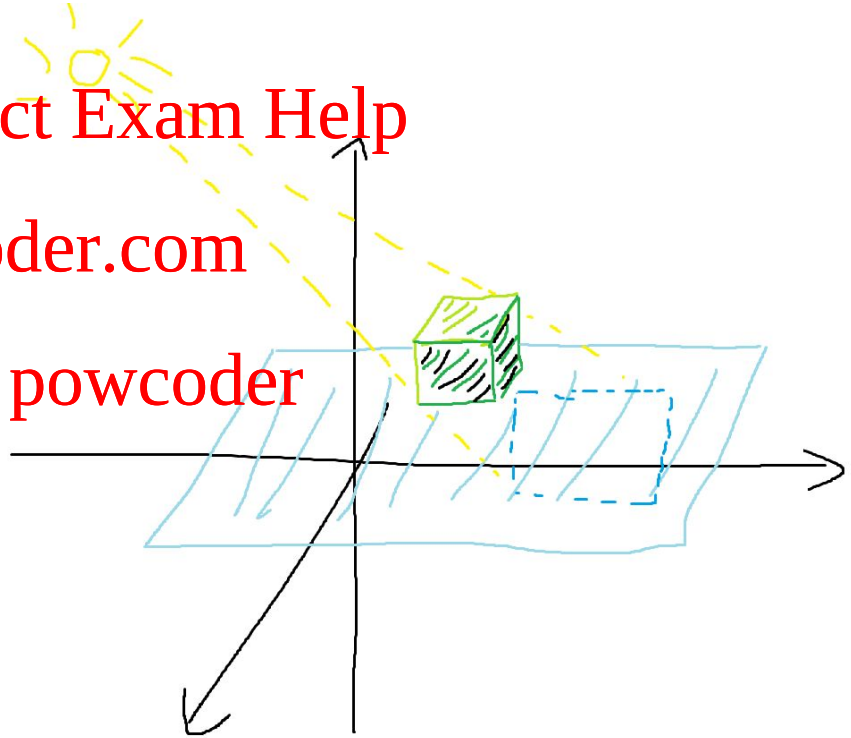
- Post processing lighting
- Lights as geometry

Shadow Mapping

Blinn-Phong Lighting and Shadows

What's our current system?

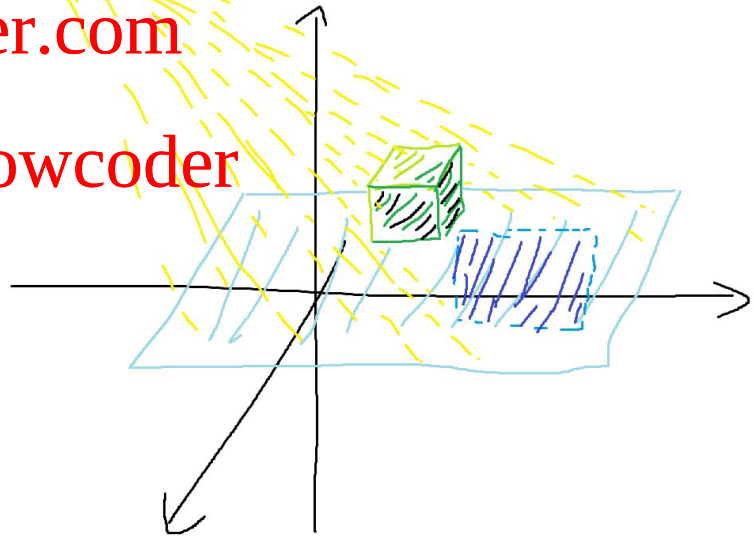
- Calculating fragments based on angle of light and viewer
- "Bottom" surfaces of objects correctly receive no light
- No detection of occluding objects!
- Surfaces light shouldn't reach will still be lit



What Would Ray Tracing Do?

Light through rays rather than angle calculations

- Any rays that hit an intervening object
- . . . will not reach other objects
- Ray Tracing is still expensive
- Something more efficient?
- Use our current rendering tech?



What are we missing?

Information needed for shadows

- Blinn-Phong is missing detection of intervening objects
- Ray Tracing would offer collision detection of light rays
- The crucial information:

- *Does the light reach specific objects?*

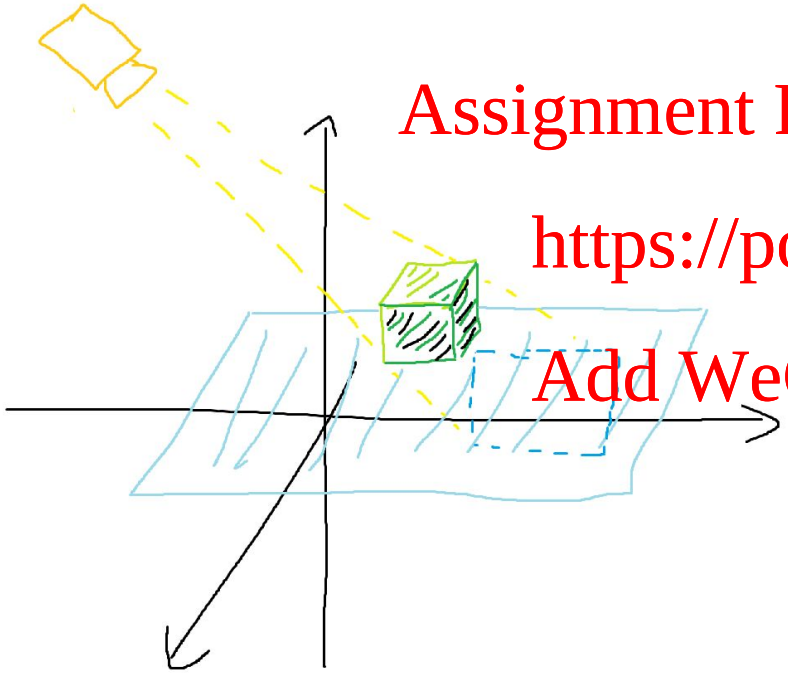
- What technique can we use to find out how far the light rays can reach?

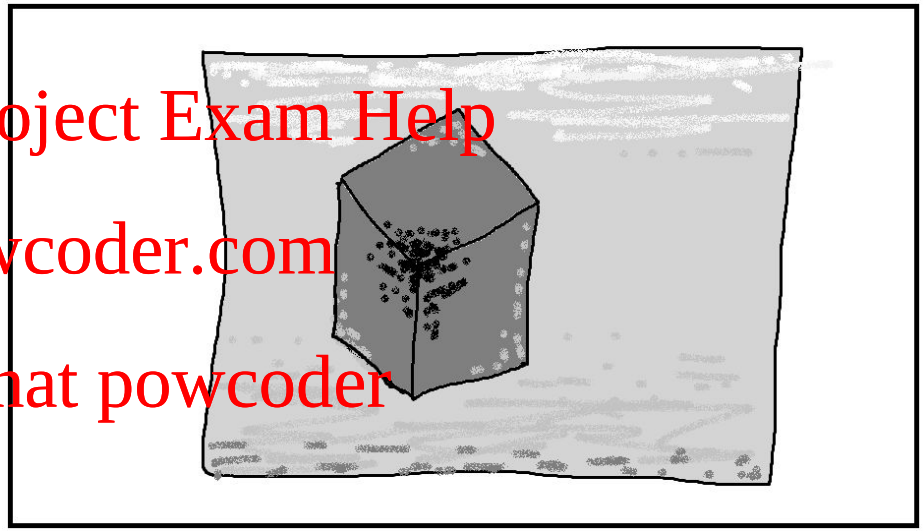
Render to a Framebuffer

A common work around to replace Ray Tracing

- When we render, we use a depth buffer
- It tells us how far away the closest visible objects are
- We want to know whether light is reaching objects . . .
- Render to a depth buffer from the perspective of the light!

Rendering from the Light's Perspective





Putting a camera where the light source is . . .
We might see a depth map that looks like this
(Darker is closer, Brighter further away)

A Depth Map from the Light Source

What Information do we get from this?

- A depth buffer will show us the closest visible fragments to the viewer
- From the light source, these are the closest fragments to the light
- The depth buffer records where the light reaches!
- We record this depth buffer to a texture called the Depth Map
- How do we use this while rendering from our main camera though?

Remember Model/View/Projection Transforms?

Transforming between perspectives

- While rendering a fragment in the main camera . . .
- Transform that fragment position into the light's view

- Sample from the Depth Map for the fragment's position
- Compare the two depth values:
  - Depth Map
  - Current Fragment Depth

- If the Depth Map sample is lower than the Fragment Depth, the light has not reached the fragment!

Depth Map Testing

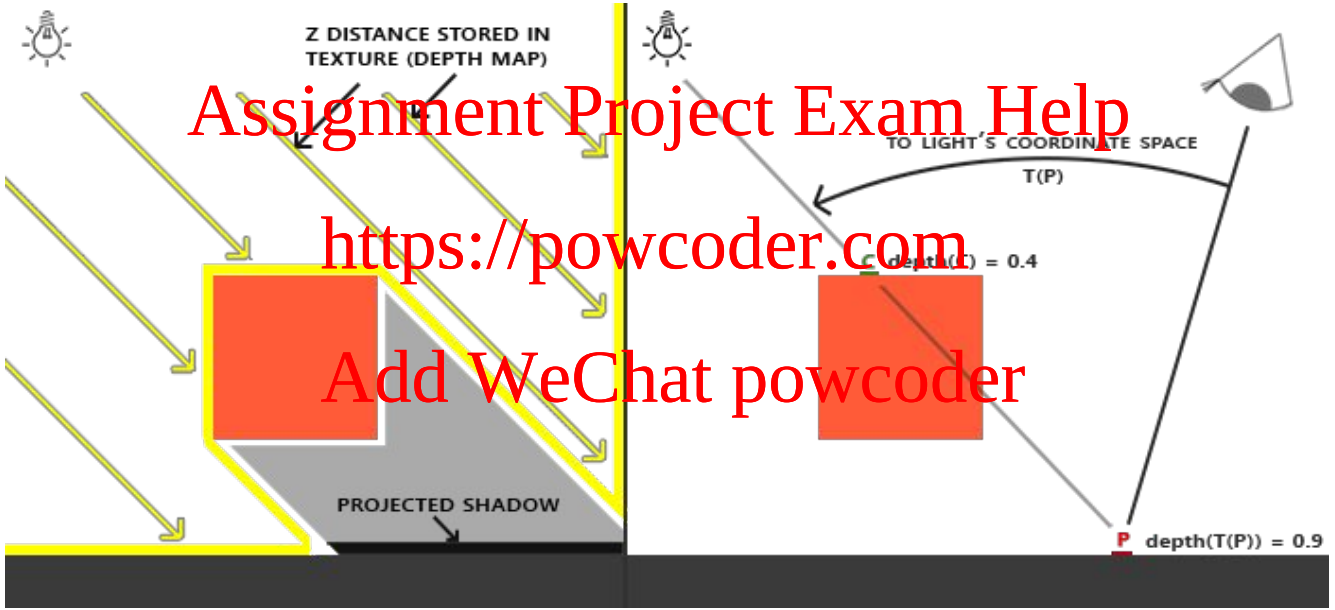


Image credit: learnopengl.com

Shadow Mapping

The process

- Render to separate depth maps for every light in the scene
- Prepare a View Transform matrix for every light
- When rendering a fragment:

  - Find the fragment position
  - For each light:

    - Transform the position into the light's perspective
    - Check depth against the light's depth map
    - If it fails the test, remove the light from the lighting equation
  - Calculate lighting for the fragment using whatever lights remain

Analysis of Shadow Mapping

Pros

- Reasonably accurate model of what light can reach
- Efficient in comparison to Ray Tracing

Cons

- Rendering the scene an extra time per light (but it's a simple depth render, not colour at least)
- Is your Depth map accurate enough? Are there sampling artifacts?
- How wide can you render? Enough for 360° point light?

Shadow Acne

I don't know where this name came from!

- An artifact of depth mapping
- Depth maps might be on an angle
- Fragments sampling the depth map might not get a perfectly correct depth
- Surfaces are able to cast shadows on themselves!

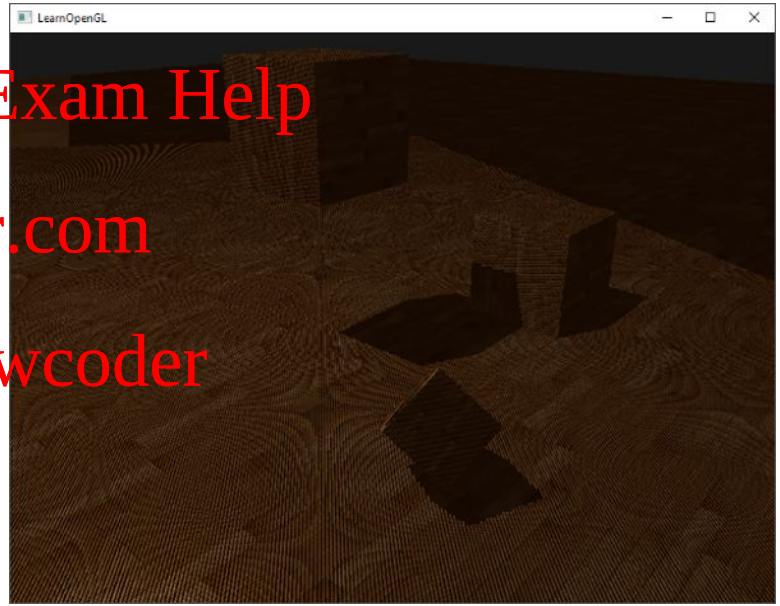


Image credit: learnopengl.com

Depth Map Accuracy

Depth Maps have their own texels

- Depth measurements then have an "area", however small
- An exact depth map can have inaccuracies between:
  - The part of the depth map texel that is sampled
  - and the actual surface depth
- A surface in full light can cast shadow on itself

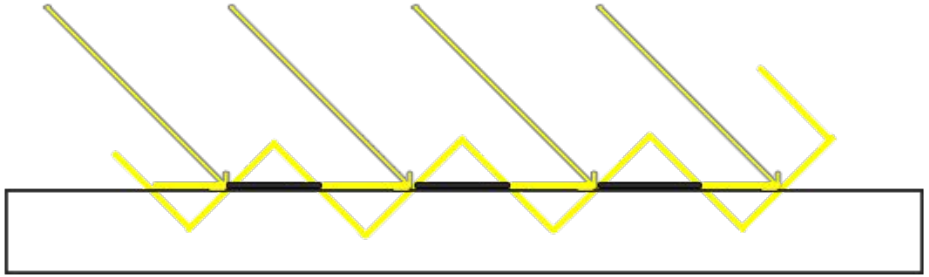


Image credit: learnopengl.com

Shadow Bias

We can introduce a Shadow Bias to correct this

- During shadow calculation
- Move the depth map "into" the object
- Then the depth map never looks like the object is obscuring itself
- This is a partial solution that can lead to "Peter Pan-ing" where the shadow detaches from the object!

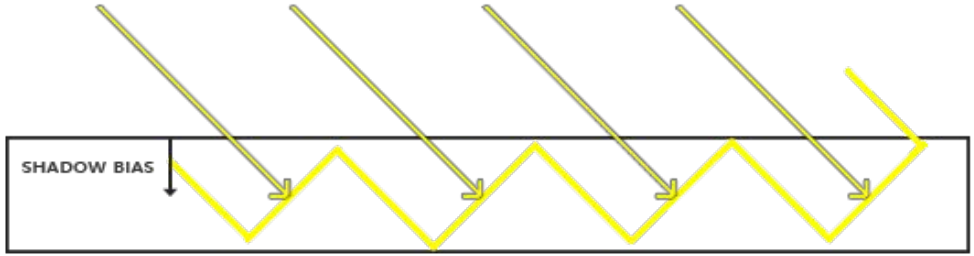
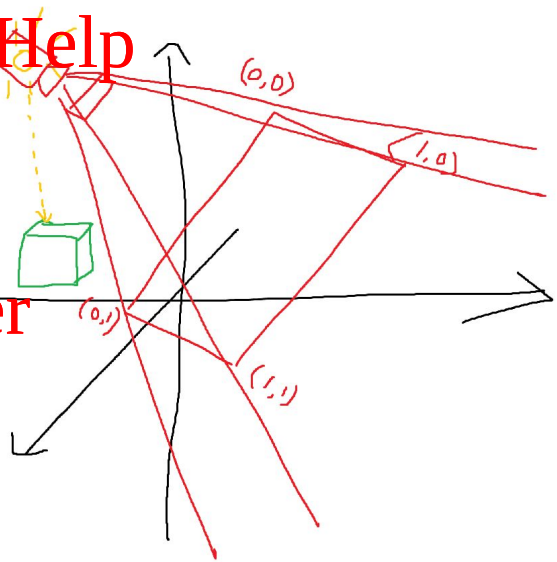


Image credit: learnopengl.com

Over Sampling

Sampling outside the Depth Map

- Like all textures, the Depth map has 0.0 - 1.0 UV coordinates
- What does this represent?
- This point light can shine on the green cube
- But the object is outside its shadow mapped frustum!
- What sampling settings should we use here?



Correcting Over Sampling

We can't cast shadows if we can't detect obscuring objects

- Anything outside our light's frustum should be lit
- What value in our depth map keeps things lit?
- The maximum, furthest depth, 1.0

- This works, unless our object is further than the far plane of our light's frustum (higher than 1.0)

- We can detect a fragment distance higher than 1.0 though
  - If depth map >= 1.0 && fragment distance >= 1.0
  - then apply light to that fragment

More Shadows …

There's more we don't have time for

- Shadow Mapping tends to cause jagged edges
  - Perspective Aliasing
  - Projective Aliasing
- We'd want to smooth them out
- There are more advanced techniques for Shadow Bias also
- We can make shadow mapping more efficient by custom fitting the light's frustum to the scene

Break Time

A story of a Graphics Engine: CryEngine, by Crytek

- CryEngine started as an Nvidia tech demo (1999)
  - Was known for much longer possible view distances than other engines
- Far Cry (2004) with Ubisoft
  - CryEngine gains a reputation for capabilities to render large, open environments
- Crysis series (2007-2013)
  - "Can it run Crysis?" - Reputation for very steep hardware requirements
  - Well known for volumetric lighting/shadows and motion blur
  - CryEngine 3 Lighting demo (2010): https://youtu.be/vPQ3BbuYVh8
  - CryEngine 3 also pushed features like dynamic vegetation and dynamic caustics
- Eventually bought by Amazon as Lumberyard (now Open 3D Engine)

Deferred Rendering

Limitations of Forward Rendering (our current lighting)

Too many lights? Too many fragments?

- Work per fragment is multiplied by the number of lights in the scene
  - Many of these lights won't even affect the fragment
- Multiple fragments per pixel overwriting each other and wasting resources
- Inefficient algorithm that also includes wasted work when there are several lights in a scene
- O(lights * fragments)

What's the solution?

Deferred Rendering

- Defers lighting until after a fragment is confirmed to be visible
  - Uses framebuffers storing different information

- Light volumes
  - Lights are rendered as geometry, limiting which fragments they affect
  - Also treats lights much like other objects, instead of things that affect all objects

- Effectively: O(lights + pixels)

Deferred Rendering Steps

A system that only lights the visible fragment

- We do a first render pass with geometry and no lighting
- Store the information in some frame buffers
  - Each buffer is a standard screen sized, 3 float frame buffer

- Then do lighting for each pixel

The Geometry Pass

The G-Pass stores information in the G-buffer

- Four standard buffers:
- Fragment position
- Surface normal
- Albedo (diffuse colour)
- Specularity (specular colour)

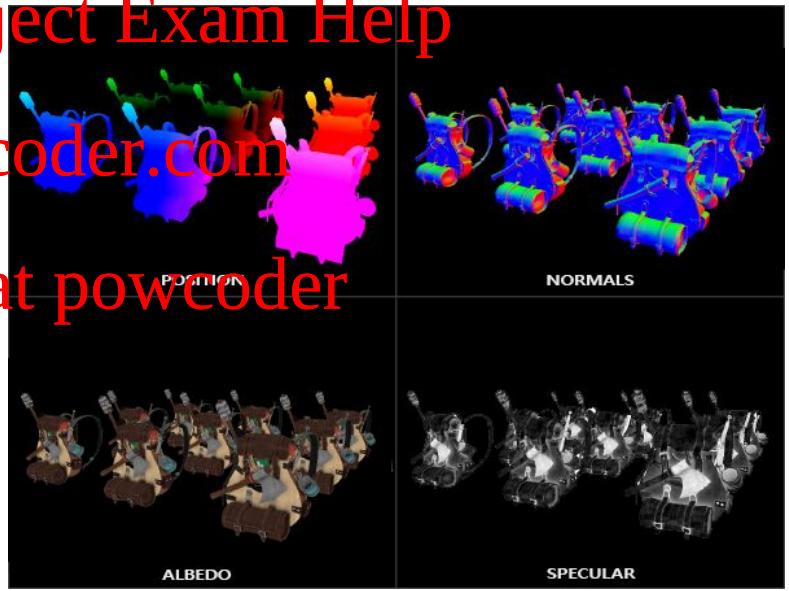


Image credit: learnopengl.com

Information per pixel

What can we do with this information?

- The Lighting Pass
- Loop through all pixels
- Calculate lighting for each pixel like it was a fragment
  - G-buffer should have all the information we need

How much has this helped?

- In complex scenes with overlapping fragments
- We've reduced the number of lit fragments to exactly the number of pixels

What's next?

Geometry Pass Issues

- We've deferred lighting
- But we're still applying every light to every pixel
- (still applying lights that might not have an effect)
- Add a technique to limit which lights apply to which pixels!

Restricting Lights

Turning off some of the lights!

- We could try to calculate where lights have effect
- Radius for attenuation of point lights
- Cones for spot lights

- But if we put this branching into the shader, our parallel GPUs have issues
- In large scale parallel calculation like GPU cores

- They all run the same code in lock step
- Which means we can't use if/else to reduce the amount of work!

Light Volumes

Rendering Lights like they're Geometry

- In the Lighting Pass:
- Render each light as a sphere (or other simple shapes like cubes)
  - Size based on attenuation radius
- Each light volume will affect certain pixels
- Calculate the light's effect on those pixels only

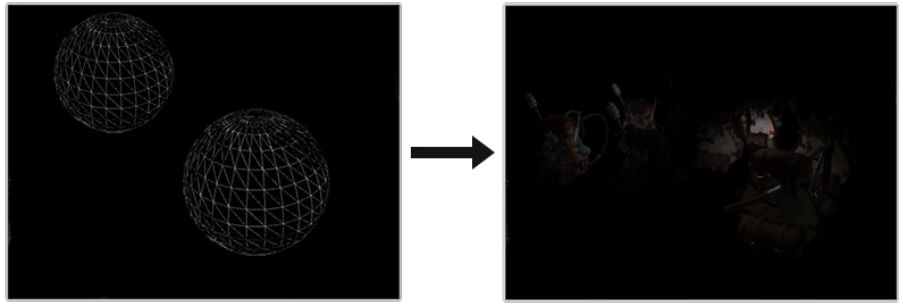


Image credit: learnopengl.com

Light Volumes Details

Some things to think about

- Rendering Light volumes lowers the number of lighting calculations
  - Each light renders a constant number of pixels, not a multiplication of fragments
- But adds a geometry render pass
  - Which is why we use simple objects
- Need to take into account the visibility of the light spheres
  - If you're inside a sphere, can you see it?
  - Need to adjust culling settings

Deferred Rendering

Pros

- Significantly reduces number of lighting calculations
  - Better the more fragments and lights there are
  - Particularly good for multiple small lights

Cons

- Extra render passes
  - G-Pass and Lighting Pass
- Memory usage
  - Storing G-buffer and light geometry
- No differentiation of objects
  - Can't use custom shaders for separate objects

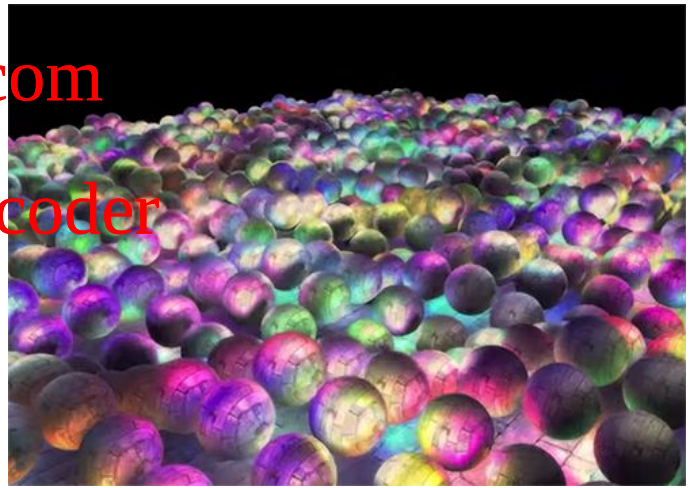


Image credit: Hannes Nevalainen
c/o learnopengl.com

What did we learn today?

Advanced Techniques using Framebuffers and Render Targets

- Shadow Mapping
  - Rendering Depth from the perspective of the light
  - Detecting intervening objects that would cast shadows
- Deferred Rendering
  - Defer rendering until after the visible fragment is decided
  - Renders all lighting specific information into a G-buffer
  - Use light volumes to limit which fragments are affected by which lights