Computer Graphics

COMP3421/9415
2021 Term 3 Lecture 16

What did we learn last lecture?

Reflections

- Cube Maps
  - Sampling via directional vectors
- Environment Mapping
  - Reflections in static environments
- Realtime Cube Maps
  - Frame Buffers
  - Render to Texture
  - Some discussion of efficiency in realtime reflections

What are we covering today?

More about Reflections

- Reflections from planes

Continuing using framebuffers

- Post Processing
- Screen Space Effects

Sphere Maps

By request: Spheres vs Cubes

- A Sphere map is a single texture
- Represents most directions around an object
- Sampling the texture via inversion of reflection direction to UV coordinates

Image credit:
https://www.pauldebevec.com/

Sphere Map Creation

The mirror sphere idea

- Can be created by taking a photo of a spherical mirror
- Also a direct mapping between sphere's normals and texels
- Creation can use the same maths to write to texels

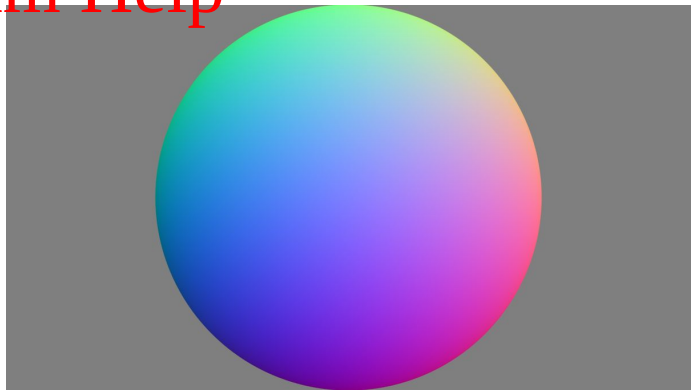


Image credit: ShaderToy user Zavie
(https://www.shadertoy.com/view/XsfXDr)

Sphere Map Analysis

Pros

- Fits on one texture

Cons

- Doesn't actually use all the texture memory assigned to it
- Loses detail around the edges (angles closer to 180°)
- Viewpoint dependent (hard to reuse if the camera moves)
- Sampling is a little bit more involved than cube maps
- Linear Interpolation gives slightly incorrect results

Planar Reflections

Mirrors and Water

Direct Reflections from flat surfaces

- We've covered arbitrary reflections in many directions
- We could just use our cube map reflections
- But surely it's simpler than that!

Ray Tracing!

- In some newer cases, yes, realtime ray tracing is definitely used!
- But we'll also look at a lower complexity technique

Learning From Tricks

Back to the Duke Nukem Example

- A mirrored copy of the scene
- Created in entirety with complete geometry

Using the idea

- We're not doubling up on geometry
- But can we use vector maths
- And framebuffers and render targets
- To "reflect" our viewpoint

Image credit: 3DRealms and Gearbox Software

Framebuffers and Render Targets

We used these to make cube maps

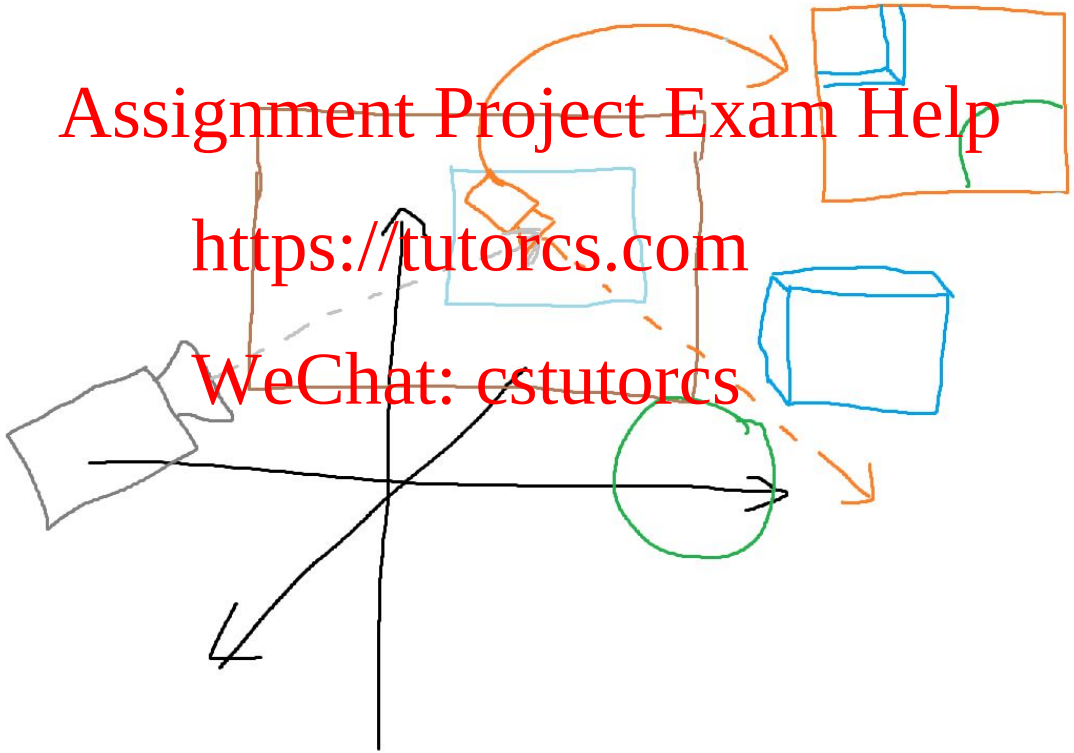
- The plane's surface is the framebuffer
- The angle of reflection gives us the camera's view angle
- We can do a second render of the world from the plane's perspective!
- We can do a single render without a cube map
- This only works because all the reflected vectors are roughly the same

The Mirror Camera Setup

A simple way to implement a reflective plane

- Place a camera at the centre of a mirror
- The mirror's texture is a render target from that camera
- Sync that camera to the main camera
  - Up vectors are the same
  - LookAt vector is reflected based on the mirror's surface normal
- Render the scene from the mirror camera
- Render the scene from the main camera, using the new texture on the mirror

The Mirror Camera Setup



Analysis of the Simple Mirror Camera

Pros

- Roughly correct
- Second render is faster than six renders for a cube map

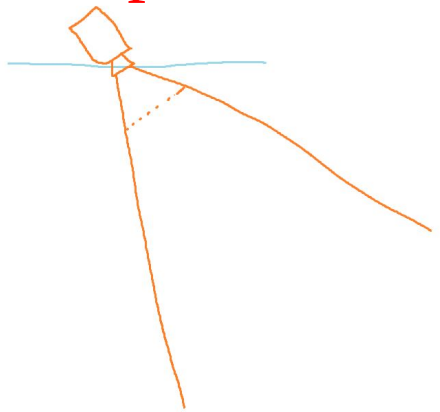
Cons

- How many mirrors do you have?
  - Every mirror in the scene needs its own setup
- Camera Location/Near Plane issues
- Is this perspective exactly correct?

Camera Location/Near Plane Issues

A top down view of the mirror camera

- Where is the near plane of the camera?
- Is it too far from the mirror?
  - Close objects aren't rendered
- Is it clipping through the mirror?
  - Might render the back of the mirror

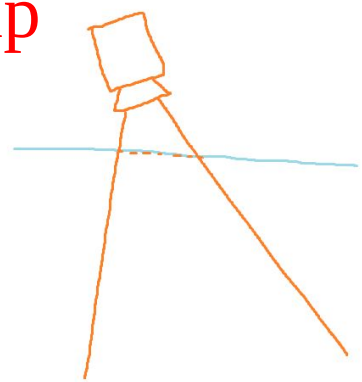


A simple camera at the mirror

Near Plane Correction

Modify the Near Plane?

- What about a camera behind the mirror
  - With a modified near plane
- Modifying the near plane
  - Custom clipping plane
  - Modification of projection matrix



A camera behind the mirror with a modified near plane

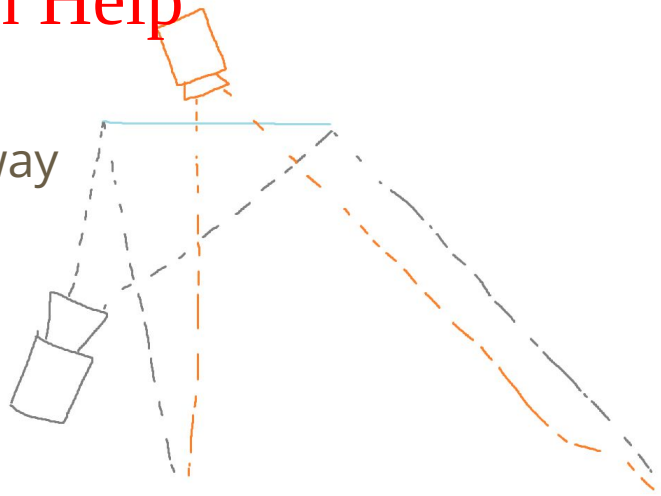
Perspective issues

Does this mirror look right under close inspection?

- Under scrutiny, the perspective is strange
- A camera at the mirror
- The main camera is potentially much further away
- Their frustums are not equal!

How do we correct this?

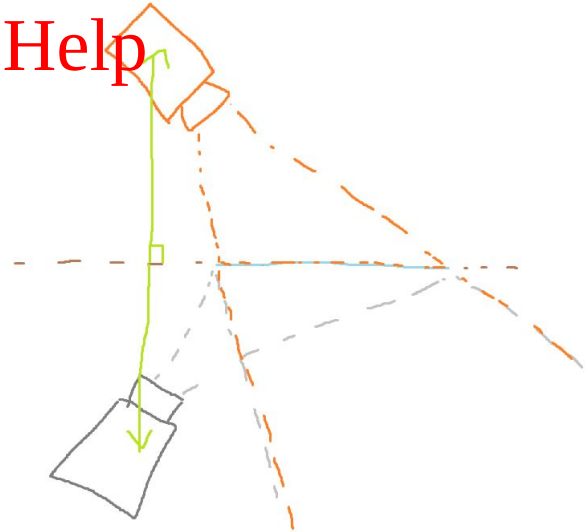
- (this time the answer isn't ray tracing!)



Perspective Correction

Move the mirror camera

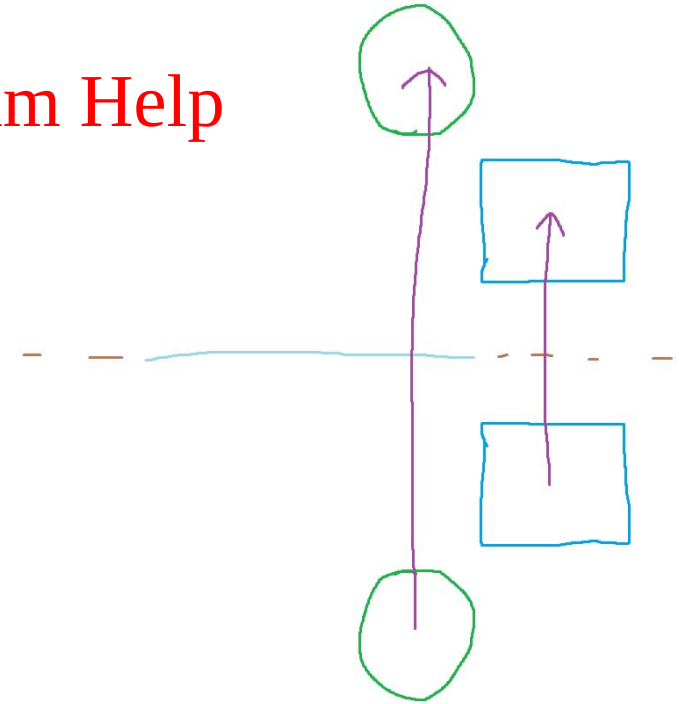
- Let's upgrade the reflection
- Not just reflect the direction of the main camera
- But reflect its position also!
- Remember the near plane needs to be modified or replaced by a culling plane at the mirror



Reflection without an extra camera

Can we do this in a single render pass?

- Don't reflect the camera, reflect the world
  - (There is no spoon)
- Create a "copy" of the scene on the other side of the mirror
- We can use transforms to reflect objects
- Don't render the mirror (or render it as a transparent object)



Analysis of "Transform" Reflection

Pros

- Single Render pass

Cons

- Are your lights reflected also?
  - Are they spilling extra light into your main scene?
- How are you handling lighting on the other side of the mirror?
  - Are your directional lights still in the right direction?
- What's behind the mirror?
  - If there's another room there, did you just reflect its objects in front of the mirror?

The rippling lake

Planar Reflections with Normal Maps

- What do we do if the reflective surface isn't perfectly flat?
- RAY TRACING! (I'm joking, but it's also true)
- Again, Ray Tracing 100% works but is expensive

Without Ray Tracing?

- Simple techniques using normals to offset sampling

Normal Mapping with Planar Reflections

A simple approximation

- Generate the reflection texture for the reflecting plane
- Sample the normal map of the plane first
- Use the direction of the normals to alter the texture coordinates
  - This is calculated estimation, accuracy isn't perfect

- Sample from a slightly different position in the texture
  - Careful about sampling outside 0.0 - 1.0

Reflecting on Planar Reflections

There's a reason why mirrors are rare in games

- Generally, the 2nd camera technique saw a lot of use
- Nowadays, being replaced by ray tracing
- A question: *"Is that one mirror worth halving your frame rate?"*
  - Most games in the era from late 1990s to late 2010s said no

Break Time

Homework

- It's been a while since we gave out any "homework"
- The Abyss (1989) and Terminator 2 (1991)
  - CG in films, particularly reflective liquid
- Half Life 2: The Lost Coast (2005)
  - Valve implemented HDR with bloom and lens flare, Cube Map reflections and Refraction
- Grand Theft Auto series (1997 - 2013)
  - Witness the growth of graphics technology over more than a decade

Image credit: 20th Century Fox

Images credit: Rockstar Games

Cameras and Portals

Cameras and Render Textures

More than just mirrors

- We can place a camera anywhere in our scene
  - And orient it in realtime!
- That camera renders to a texture
- We can map that texture to any object in our scene!
- This gives us realtime security cameras, portals and other fun toys

Image credit: Valve

Post Processing

Framebuffers and Render to Texture

This technique has seen a lot of use in the last few years

- At its core:
- Render the scene to a framebuffer (the same size as the screen/window)
- Modify what's in that buffer

- Write the final result to the main framebuffer

- Since the work is done after the rendering is finished . . .
- . . . this is called "Post Processing"

Simple Post Processing

We can process every pixel in a framebuffer

- Read the colour data
- Write new colour data to the main framebuffer

A simple example: Black and White filter

- Read the RGB values
- Average them
- Write the same value to all three RGBs in the framebuffer

Other Simple Post Processing Effects

What else can we do while manipulating screen colours?

- Night Vision Mode
  - Green tint everything
  - Alter the intensity curve to make things look artificial
- Inverted colours
  - Making some kind of magical opposite effect
- Blood Rage
  - Turn the edges of the screen red, fading into normal colours near the centre
  - This one uses the texture coordinates to determine whether or not something changes colour

Mixing with other effects

Head up Displays (HUD)

- HUDs are not always done with post processing
  - Often just 2D elements rendered over the scene
- A transparent HUD could be done in post
  - Take a full screen HUD texture
  - Edit the values for numbers, health bars etc
  - Blend the HUD with the frame before writing it to the main framebuffer
- Alpha blend a premade effect over part of the screen
  - Damage markings like cracked glass
  - Elemental spell effects like lightning

Image credit: Xbox Game Studios

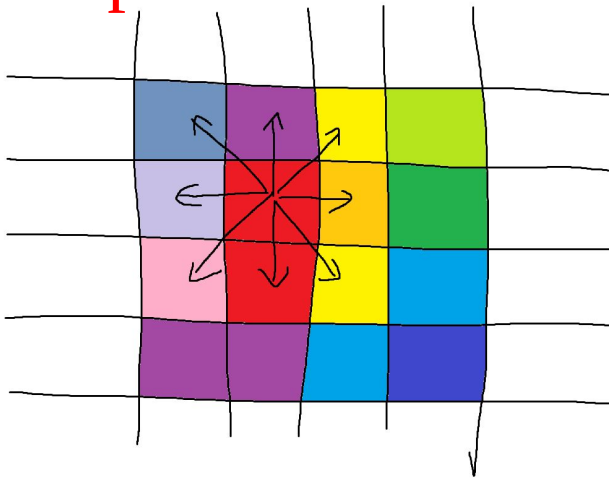
Image credit: Gearbox Software

Kernel Effects

More than just changing colours of individual pixels

- A kernel looks at the pixels around each pixel
- Usually impossible in the fragment shader
  - There's no guarantee other pixels have already been calculated
- Read the values of pixels

- Write to the current pixel based on some combination of the pixels in the kernel



A Simple Kernel Effect

Let's add a blur post processing effect

- Each pixel samples the 8 adjacent pixels
- The final colours are the sum of the kernel's calculation in each of its cells

- eg: 1/16 of the top left, ¼ of the centre

- The total is 1 to ensure that values can't sum to more than 1
- The result is each pixel being a blend of all adjacent pixels

1	2	1
2	4	2
1	2	1

/ 16

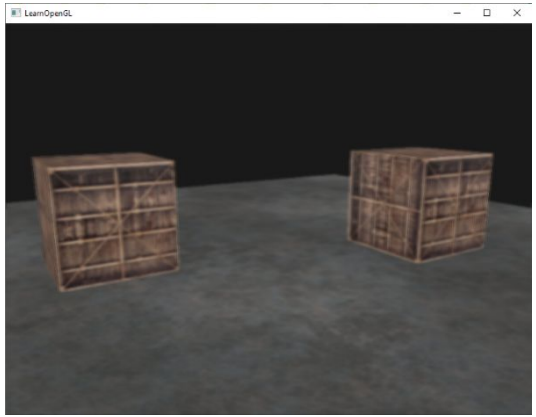
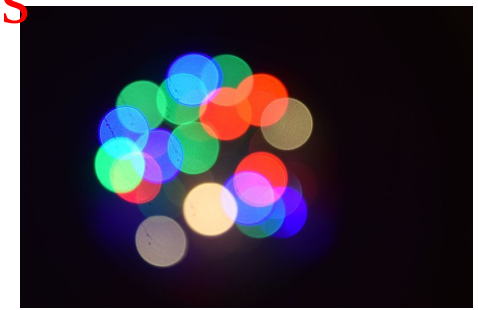


Image credit: learnopengl.com

More Complex Kernels

Different shapes!

- A kernel is not limited to the adjacent pixels
- We can sample information from more distant pixels
- And in different specific shapes
- We can do things like adding specific shaped lens flare and bokeh to our scenes
- As well as other effects



Bokeh from a physical camera
Image credit: Wikipedia user Ranjithsiji

Bloom

A complex post processing example

- Bloom is an effect that combines HDR *(lecture 13)* with post processing
- Mimics a real world effect

- Very bright objects appear larger than they are
- The light "blooms" outwards from the light source (or very bright reflection)

- Since this effect spreads light over multiple pixels, it must happen in post

HDR with Bloom

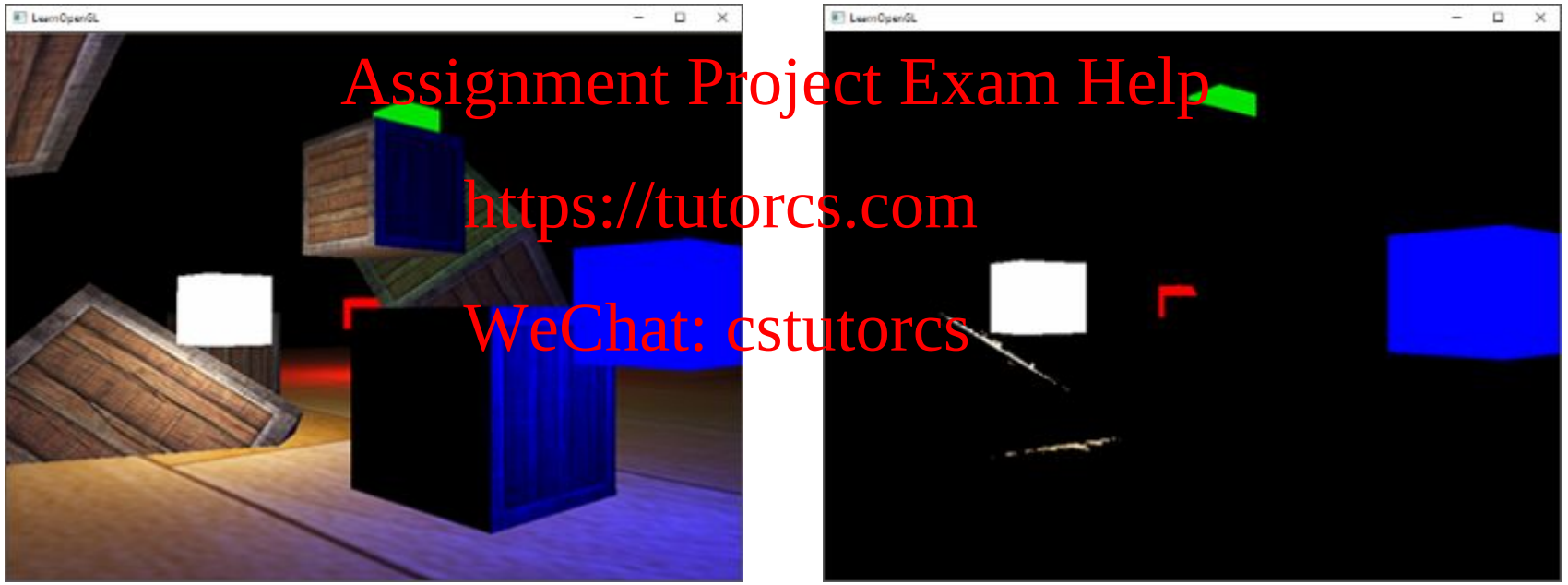
Write to the HDR Framebuffer first

- Write your light values to your floating point HDR framebuffer
- Instead of immediately applying tone mapping to reduce these values to the 0.0 - 1.0 range

- Create a new framebuffer, we'll call this the bloom buffer

- Copy only the light values that exceed 1.0 into the bloom buffer

Bloom Images



The scene on the left. The "bloom buffer" on the right
Images credit: learnopengl.com

Bleeding Light

Now we apply a blur to the bloom buffer

- We can use the blur we showed earlier
- But there are many possible kernels that will blur for different effects
- For effective bloom, we might use a Gaussian Blur

$$\begin{bmatrix} 0.00000067 & 0.00002292 & \mathbf{0.00019117} & 0.00038771 & \mathbf{0.00019117} & 0.00002292 & 0.00000067 \\ 0.00002292 & 0.00078633 & 0.00655965 & 0.01330373 & 0.00655965 & 0.00078633 & 0.00002292 \\ \mathbf{0.00019117} & 0.00655965 & 0.05472157 & 0.11098164 & 0.05472157 & 0.00655965 & \mathbf{0.00019117} \\ 0.00038771 & 0.01330373 & 0.11098164 & \mathbf{0.22508352} & 0.11098164 & 0.01330373 & 0.00038771 \\ \mathbf{0.00019117} & 0.00655965 & 0.05472157 & 0.11098164 & 0.05472157 & 0.00655965 & \mathbf{0.00019117} \\ 0.00002292 & 0.00078633 & 0.00655965 & 0.01330373 & 0.00655965 & 0.00078633 & 0.00002292 \\ 0.00000067 & 0.00002292 & \mathbf{0.00019117} & 0.00038771 & \mathbf{0.00019117} & 0.00002292 & 0.00000067 \end{bmatrix}$$

Example Gaussian Blur Kernel
Image credit: Wikipedia

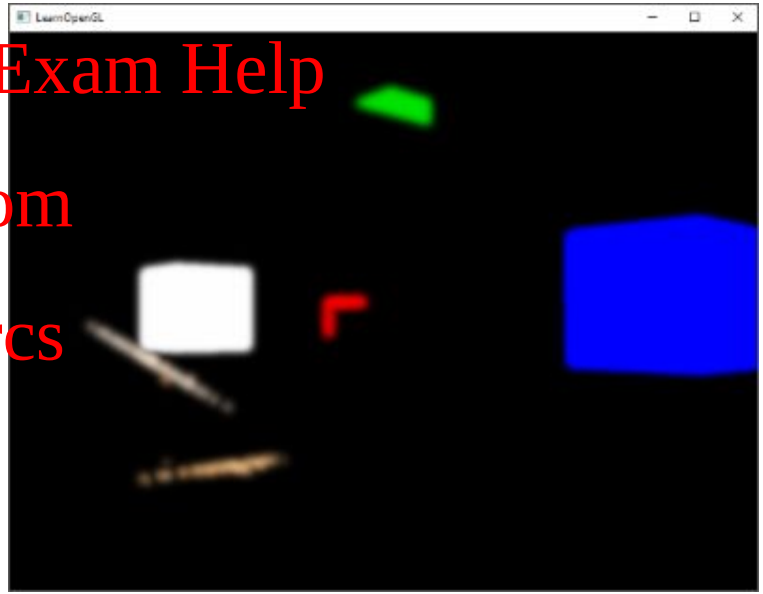


Image credit: learnopengl.com

Combine the Effect

To finalise the bloom

- We add the blurred results from the bloom buffer to the HDR framebuffer
- This makes the colour of lights expand beyond their original size
- The final scene will have any bright lights bleeding into nearby pixels

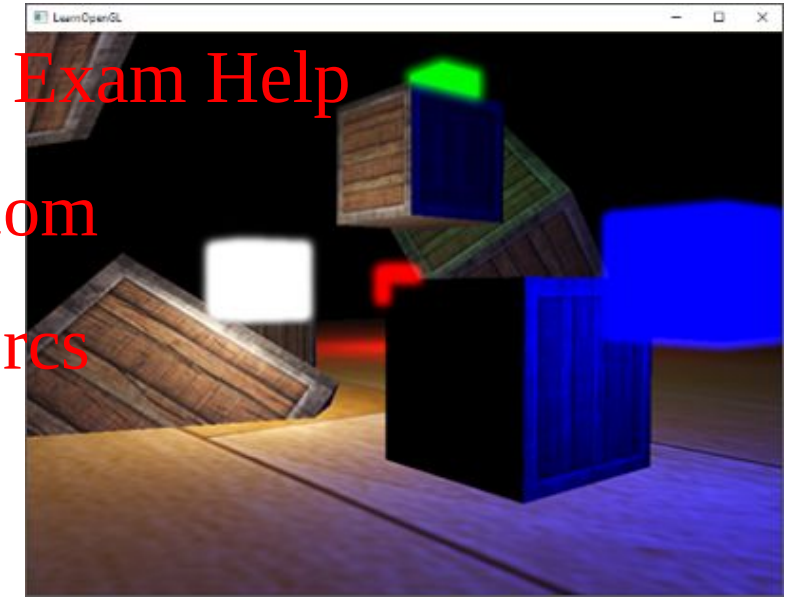


Image credit: learnopengl.com

Other post processing effects

Also sometimes referred to as Screen Space Effects

- Motion Blur
  - Saves buffers from previous frames
  - Blurs between frames, not just between pixels in the current frame
- Ambient Occlusion
  - Uses the depth buffer and surface normal
  - Darkens areas that have other geometry near them and should receive less ambient light
- Anti- Aliasing
  - Not necessarily a post processing effect, but can be implemented that way
  - Reduces jagged edges from angled lines being drawn across square pixels
- Others like Depth of Field, Colour Grading, Chromatic Aberration

What did we learn today?

Planar Reflections

- Details and conundrums of direct reflections
- Trying to calculate them efficiently
- Other uses of the technique like portals

Post Processing

- Altering the colour data after the full frame is rendered
- Using kernels to sample from nearby pixels
- Bloom as an example