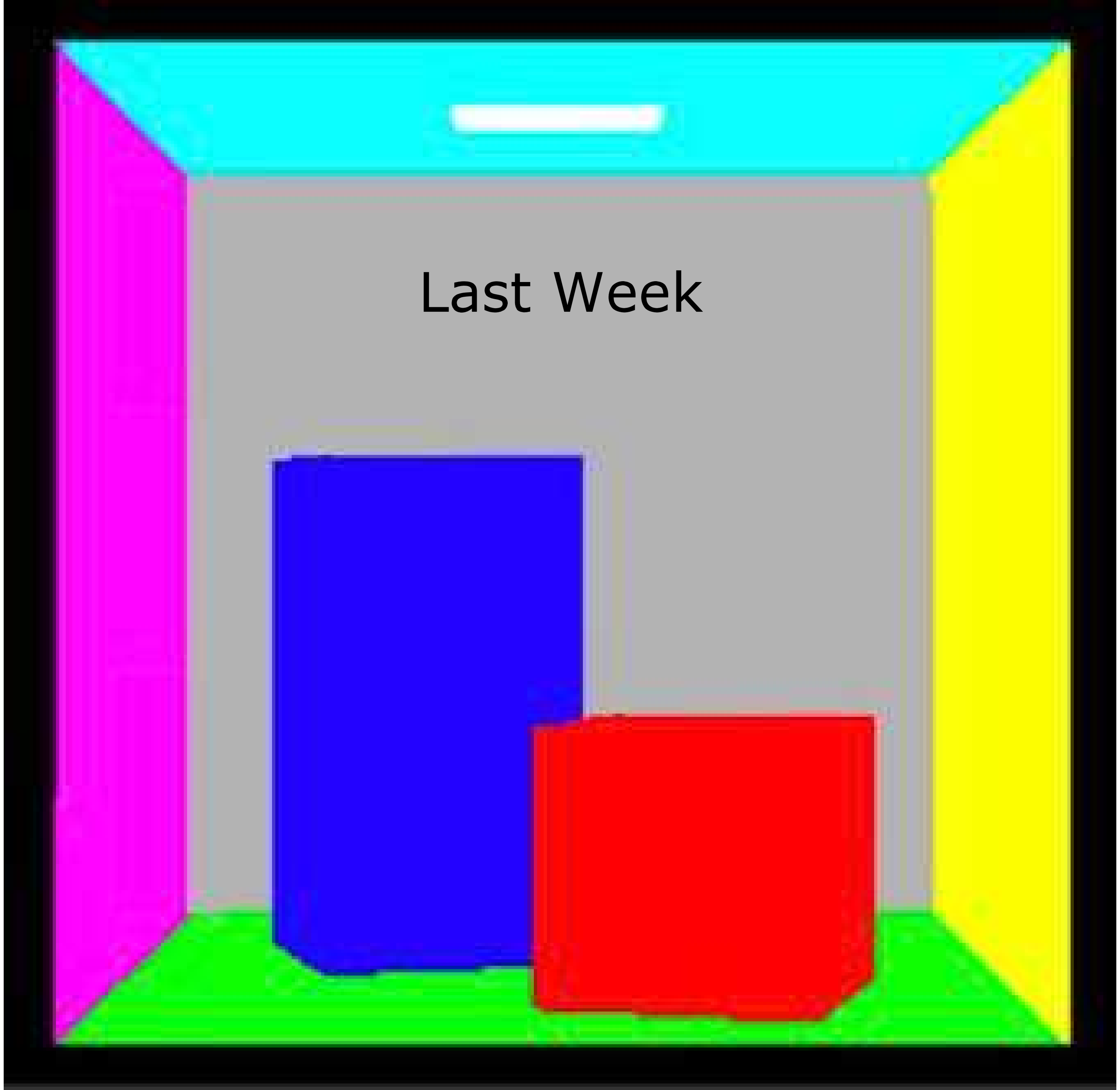
COMS30020 - Computer Graphics

Week 5 Briefing

Dr Simon Lock


Last Week

This coming week

I'm NOT saying that this coming week is easy...
However, there were a lots tasks in last workbook
Not so many (mandatory) tasks in next workbook

This was just the way the topics clustered
We put them in the most logical workbook

If you didn't finish all of last workbook
There should be a bit of slack to catch up

Coming week's topics

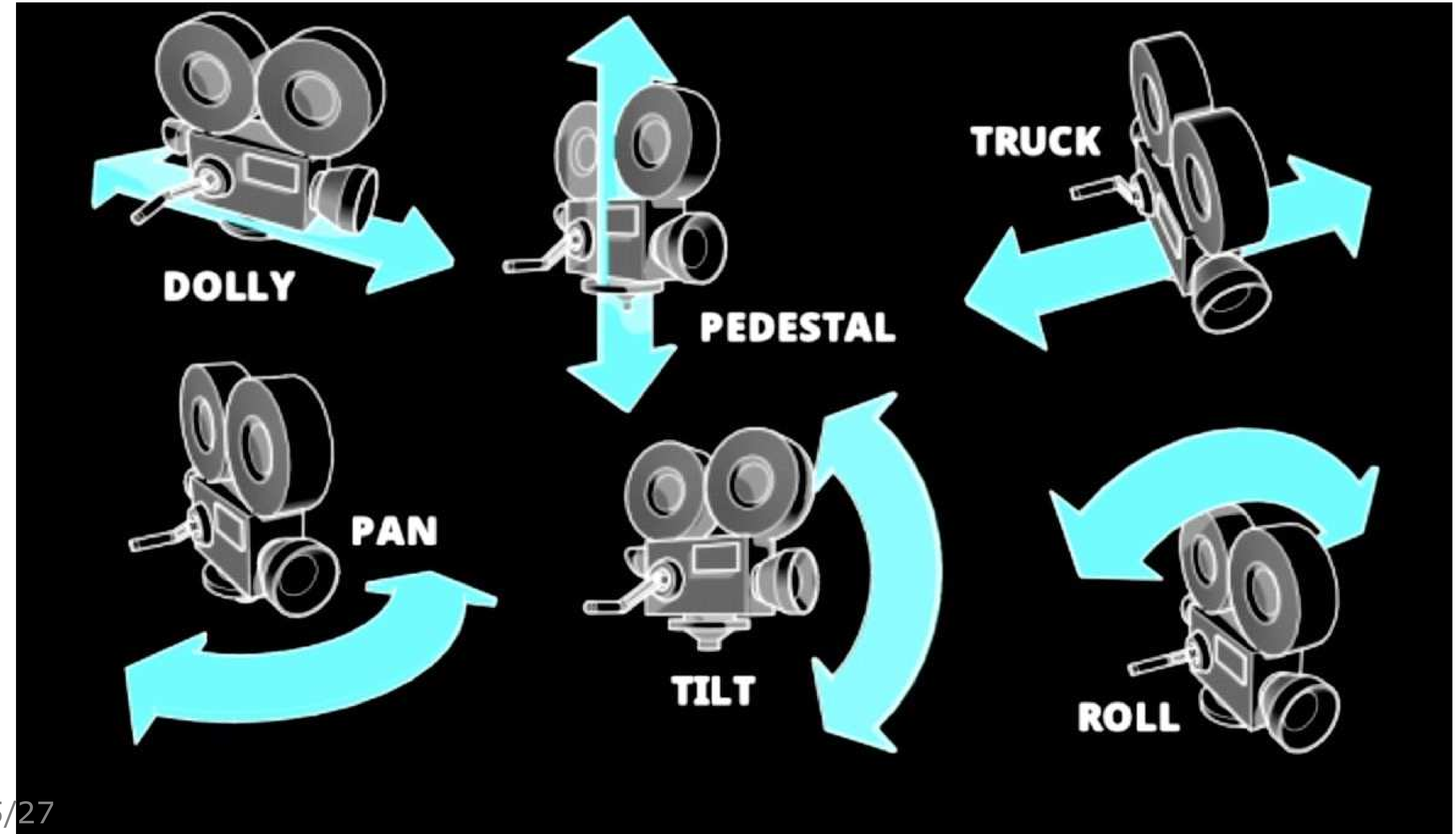
The focus of this week's workbook is "navigation"
That is: moving the camera around in 3D space

This includes translations and rotations
Also encompasses basic scripted movement

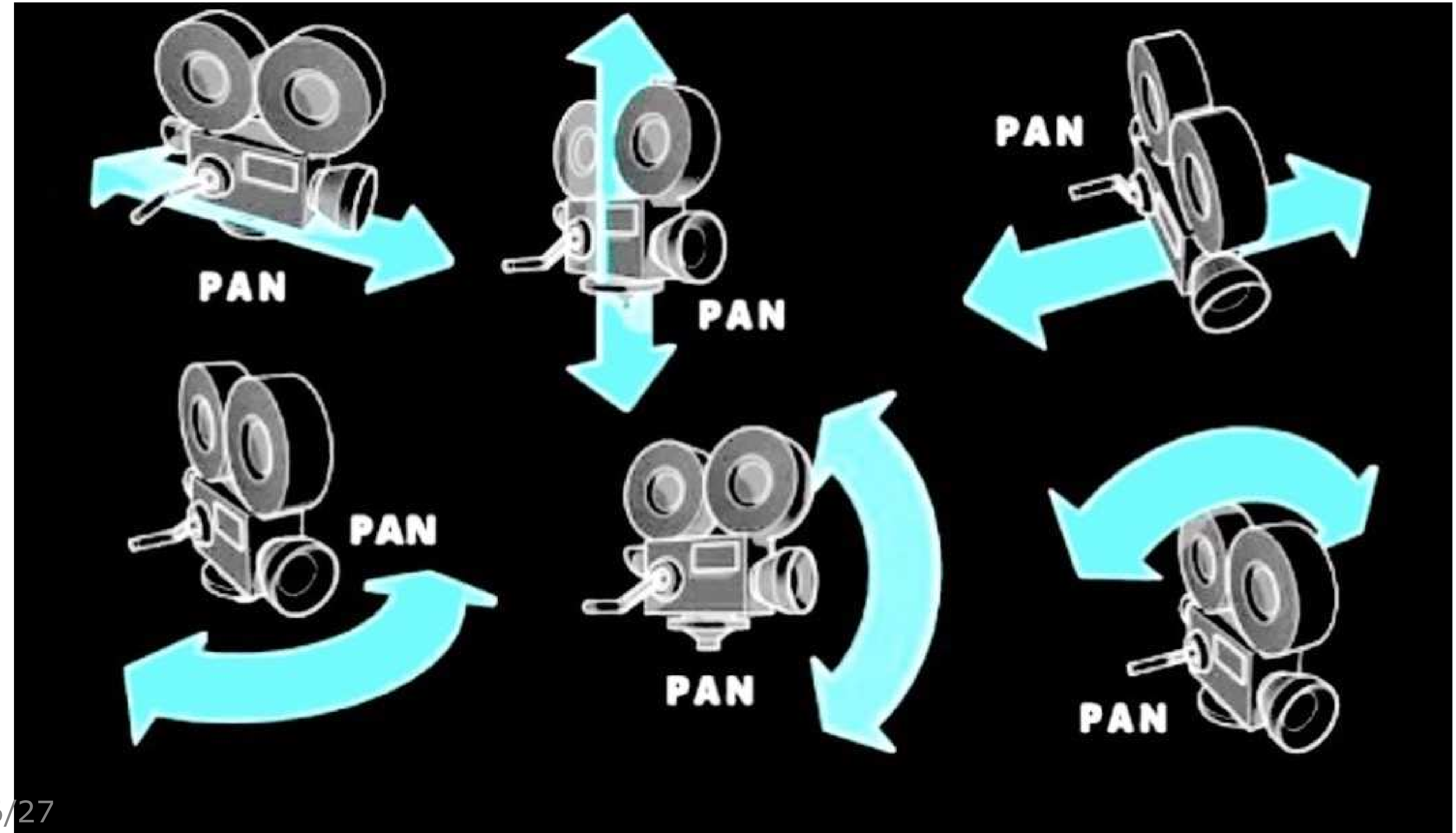
The below animation provides an overview of these:

CameraMovement

Cinematography Camera Movement



Client's guide to camera movement

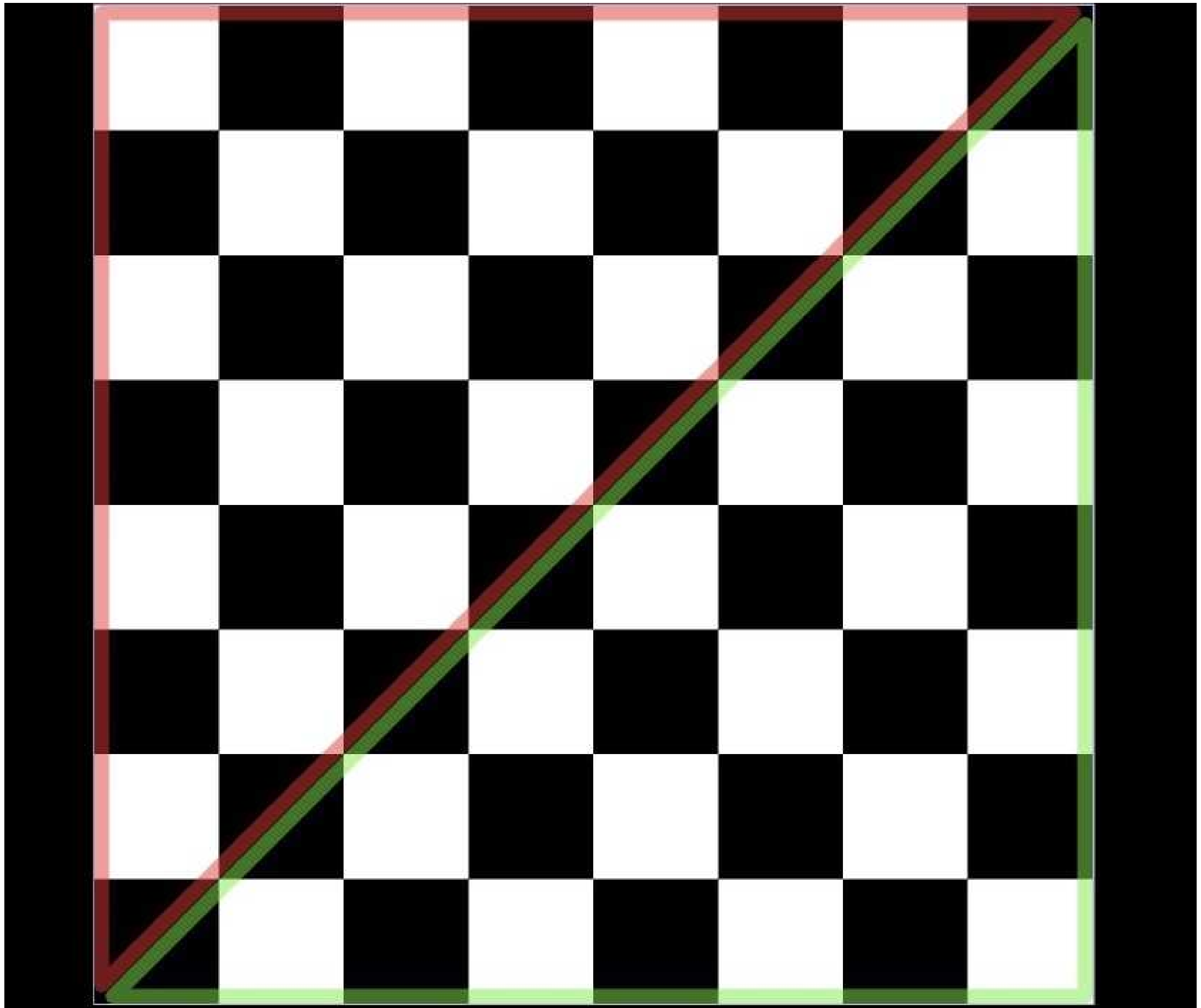


Let's take a look at the workbook:

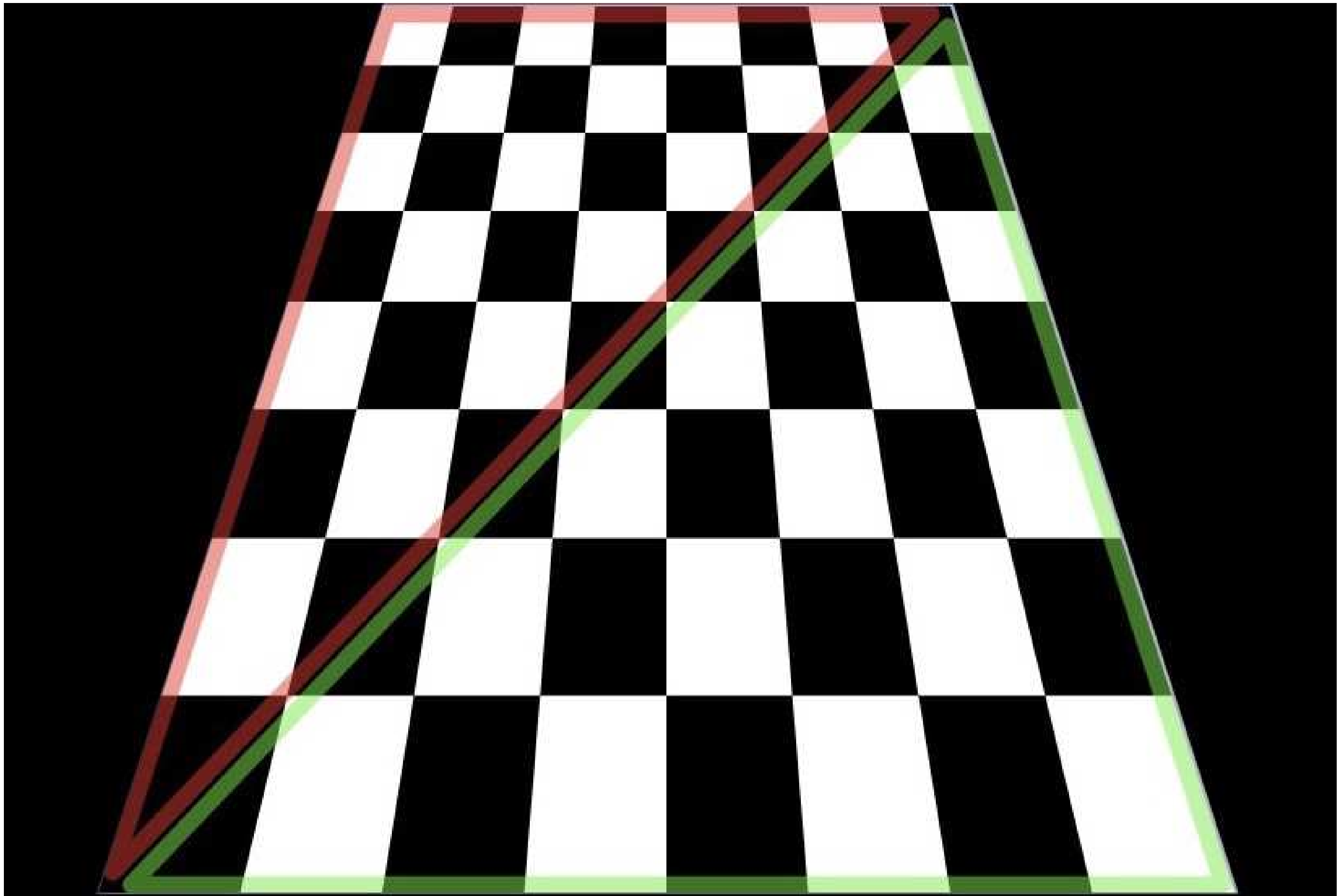
https://github.com/COMS30020/CG2023

Texture Mapping: When to stop trying...

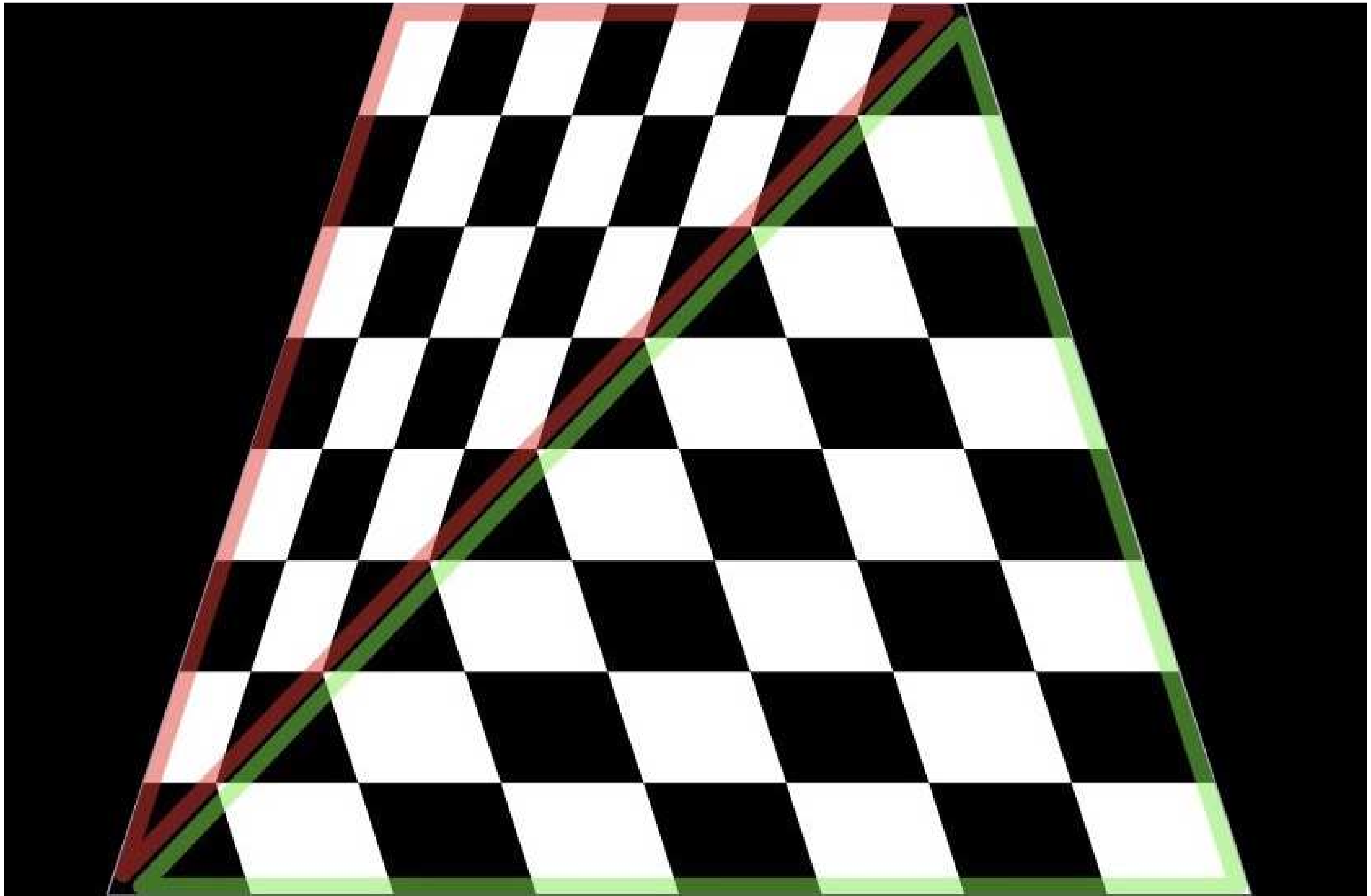
Face-on / Top-down



Correct Perspective



What your's will look like !



Homogenous Coordinates

The use of Homogenous Coordinates in optional
(moved from core material in switch to 7 week term)

It's an interesting (?) side-issue topic
But it won't gain you anything in assessment

A fair bit of work to refactor your code to vec4s
Don't even think about it unless you are up-to-date

Additional Bits of Guidance

CLion Debug Run Configurations

Some people in the lab have been asking:

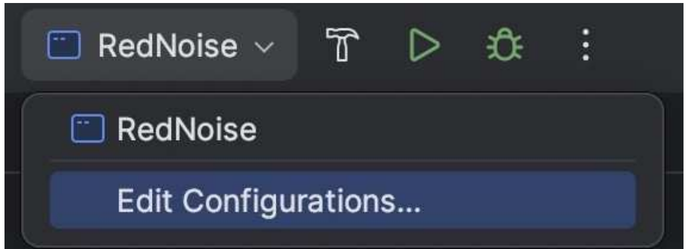
"How do you run RedNoise using the CLion debugger ?"

If you imported "CMakeLists" when you opened project

You _should_ already have a debug run configuration

If you don't have any, take a look at this tutorial:

ttps://jetbrains.com/help/clion/run-debug-configuration.htr



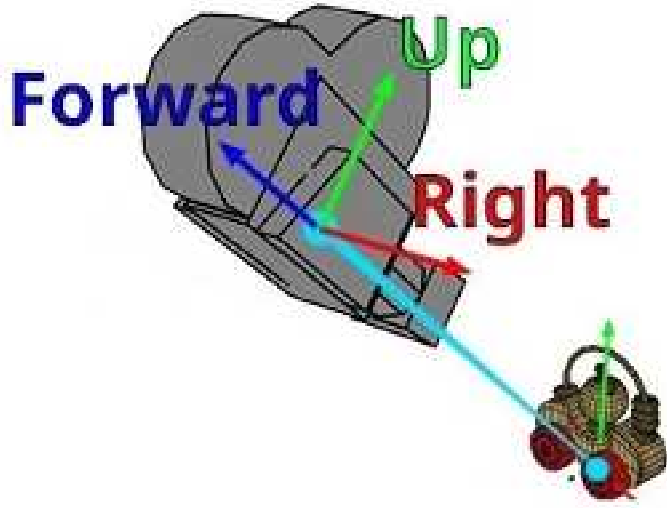
Camera Orientation

You would have _thought_ that camera orientation would be expressed in terms of *front* of camera

So why does orientation use the *back* instead ???

The reason for this is simple...
This representation makes it *easier* to apply orientation when doing vertex projection

We just multiply !



Overriden Operators

vec3 & mat3 override basic maths operators:

```
+   -   *   /
```

So you can do things like:

```
glm::vec3 from;
glm::vec3 to;
glm::vec3 difference = to - from;

glm::mat3 orientation;
glm::mat3 rotation;
glm::mat3 adjustedOrient = rotation * orientation;
```

Versatile vec3

We've discussed the versatility of the vec3 class:
Location, Direction, Colour, Intersection etc.

Just like arrays, elements can be accessed by index:

```
float red = myVec[0];
```

This can however make code hard to understand
For example, you have to remember that 2 is blue !

To aid readability, vec3s have various "aliases"...

vec3 Aliases

Colours:

```
float red = myVec.r;
float green = myVec.g;
float blue = myVec.b;
```

Positions/directions:

```
float x = myVec.x;
float y = myVec.y;
float z = myVec.z;
```

There are others - we might talk about them later

Printing out vec3s

Often we need to know the content of a vec3
We _could_ print out the elements individually
(maybe using the aliases shown previously)

However...

GLM provides a method to convert vec3s to strings

```
#include <glm/gtx/string_cast.hpp>
std::cout << glm::to_string(myVec)
```

This might save you a bit of time when debugging !

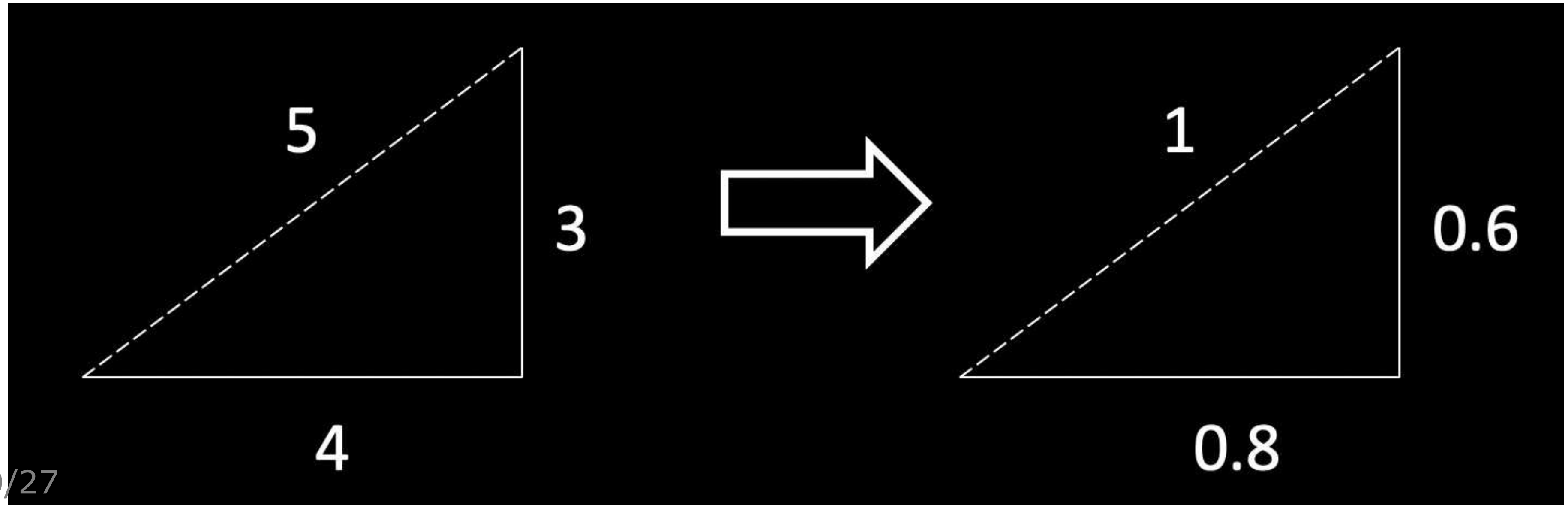
Normalisation

When you multiply things together,

If they are not unitary (magnitude of 1)

You could end up with a scaling effect !

So you may need to "normalise" vectors first



How to normalise ?

You could write your own function to normalise
But GLM already provides one, so let's use that:

```
glm::normalize(vec3)
```

Note the crazy spelling !

When to normalise ?

Golden rule:

If you NEED the distance - then DON'T normalise

Normalise at all other times !

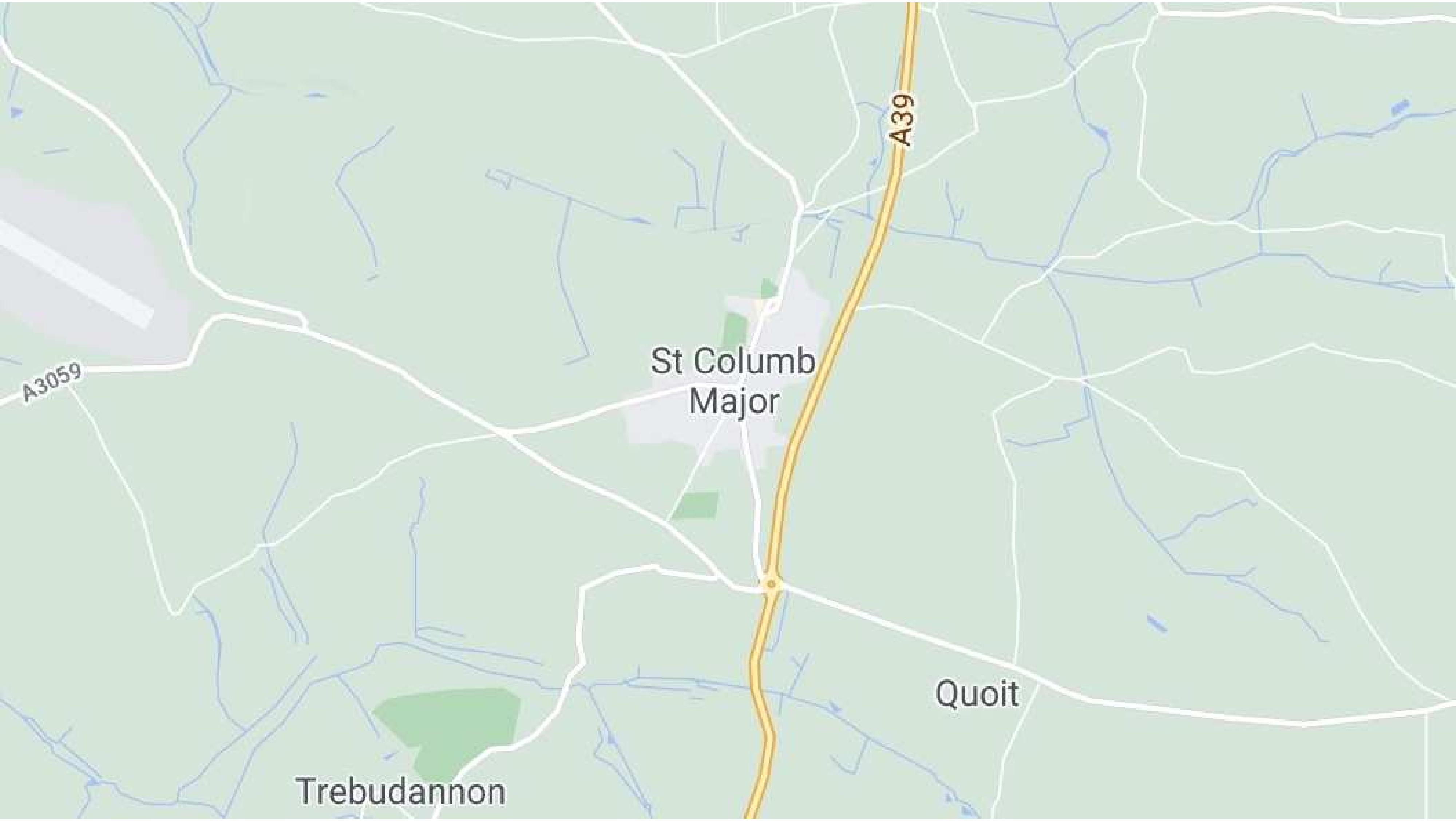
Initialising mat3s

Remember that GLM mat3s are "column major"
You initialise them with columns, rather than rows:

```
glm::vec3 colOne, colTwo, colThree;
glm::mat3 myMatrix;
myMatrix = glm::mat3(colOne, colTwo, colThree);
```

You can initialise them with 9 separate floats
But again, these will need to be in column order

The way I remember...


A39
St Columb Major
A3059
Quoit
Trebudannon

And Finally

We can use navigation and transformation
To explore numerical data using animation
Powerful tool to put across convincing arguments

2D animation is useful
3D animation _can_ be even more powerful
Here is a nice example I saw recently:

climate-spiral-fast

Finally, Finally

Navigation & Transformation in 3D models

A nice example by Michel Gondry

StarGuitar

https://www.youtube.com/watch?v=0S43IwBF0uM