COMS30020 - Computer Graphics

Week 5 Briefing

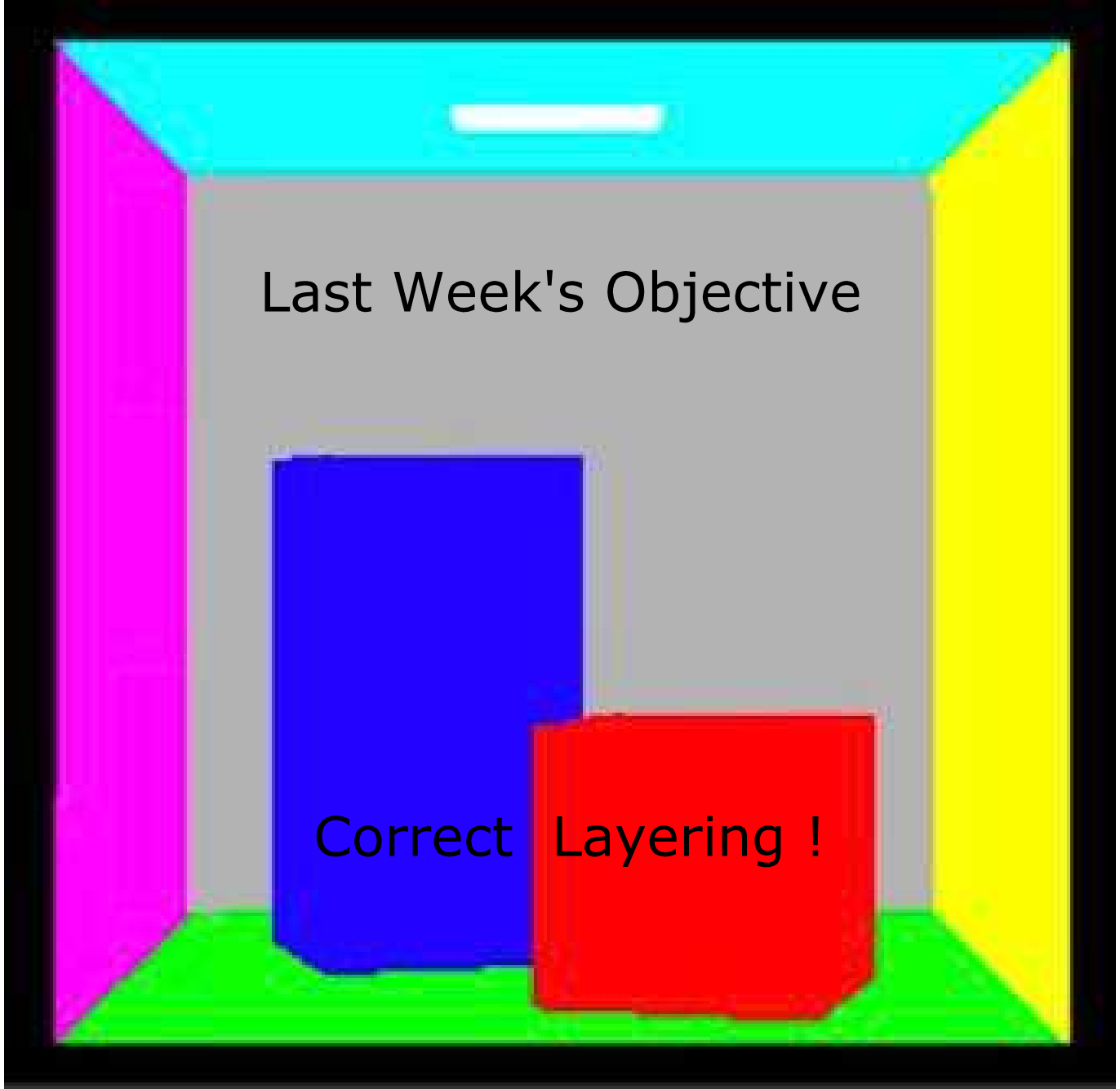
Dr Simon Lock

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 into the most logical workbook

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


Last Week's Objective
Correct Layering !

Coming week's topics

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

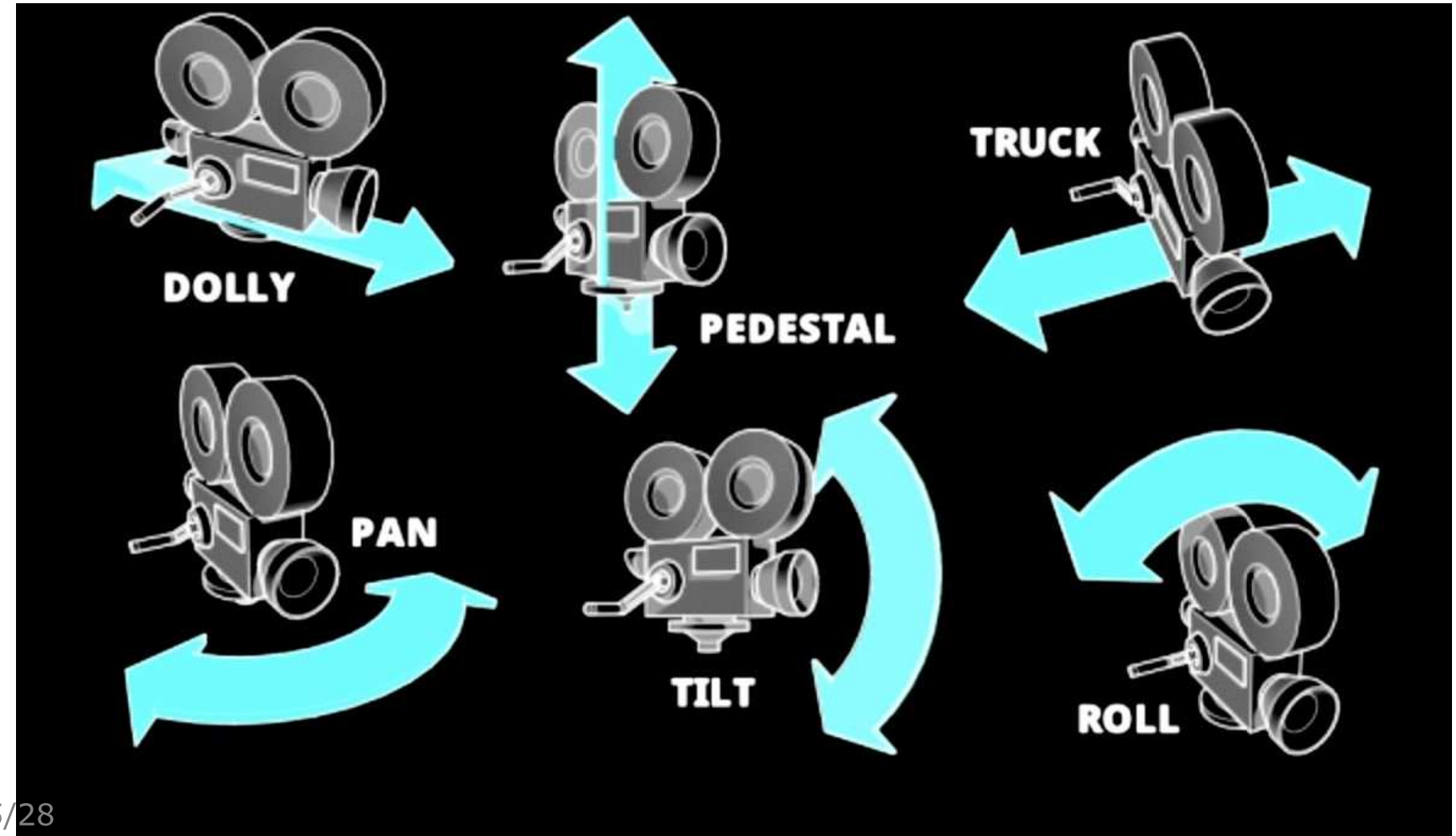
This includes basic translation and rotation
Also encompasses more complex scripted movement

The following animation provides an illustration:

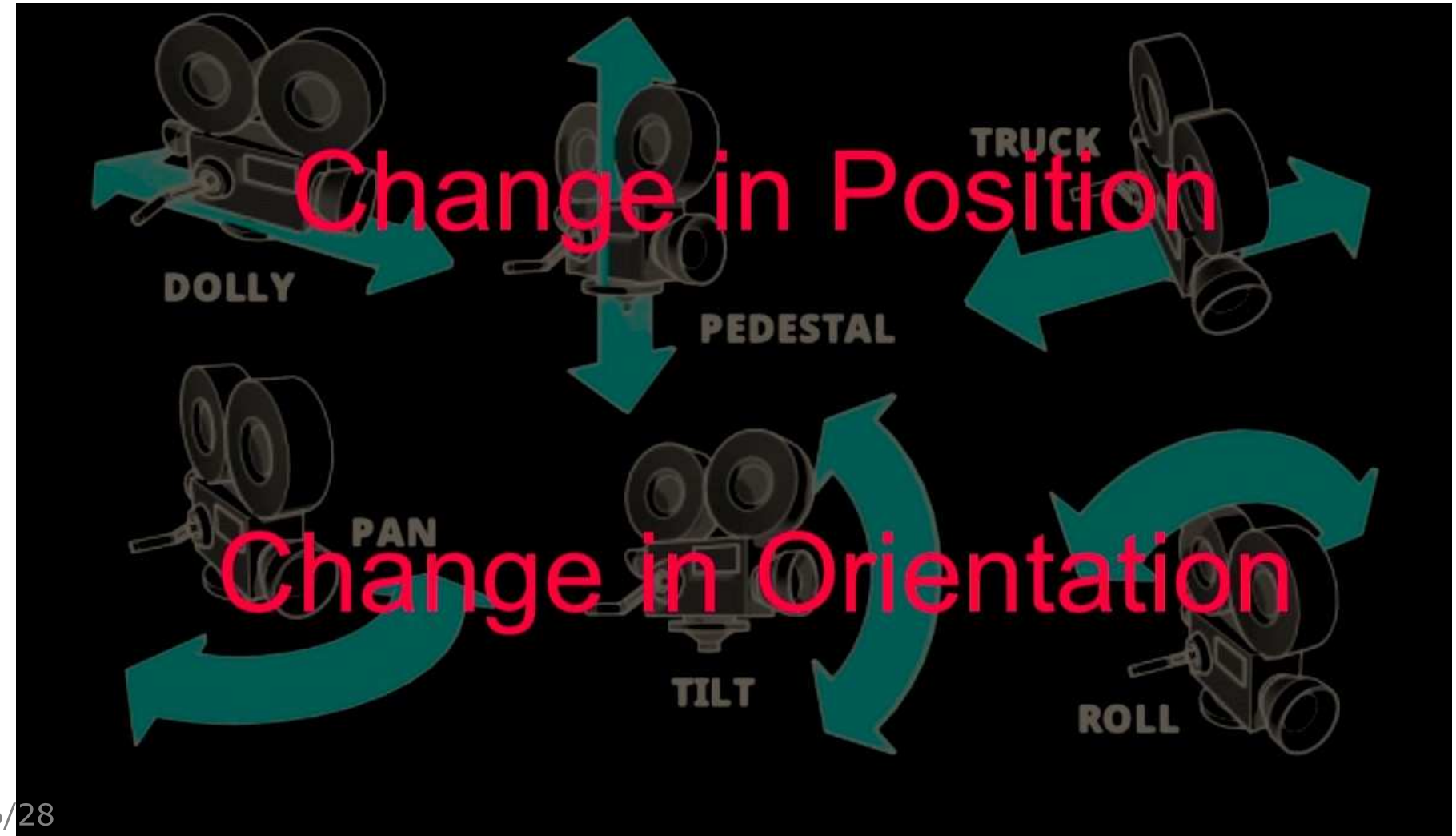
CameraMovement

Although it seems simple, people often mix them up !

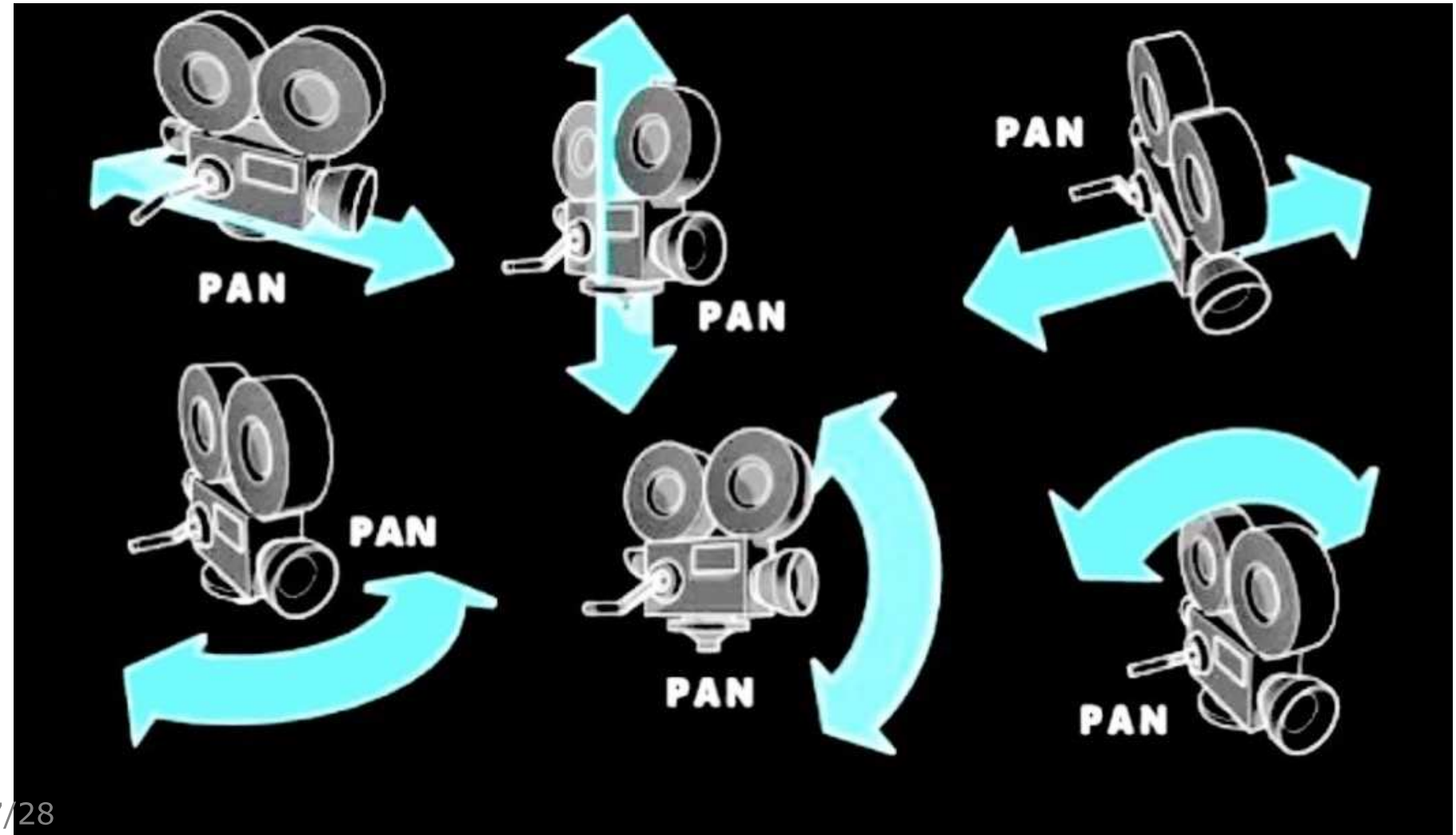
Cinematography Camera Movement



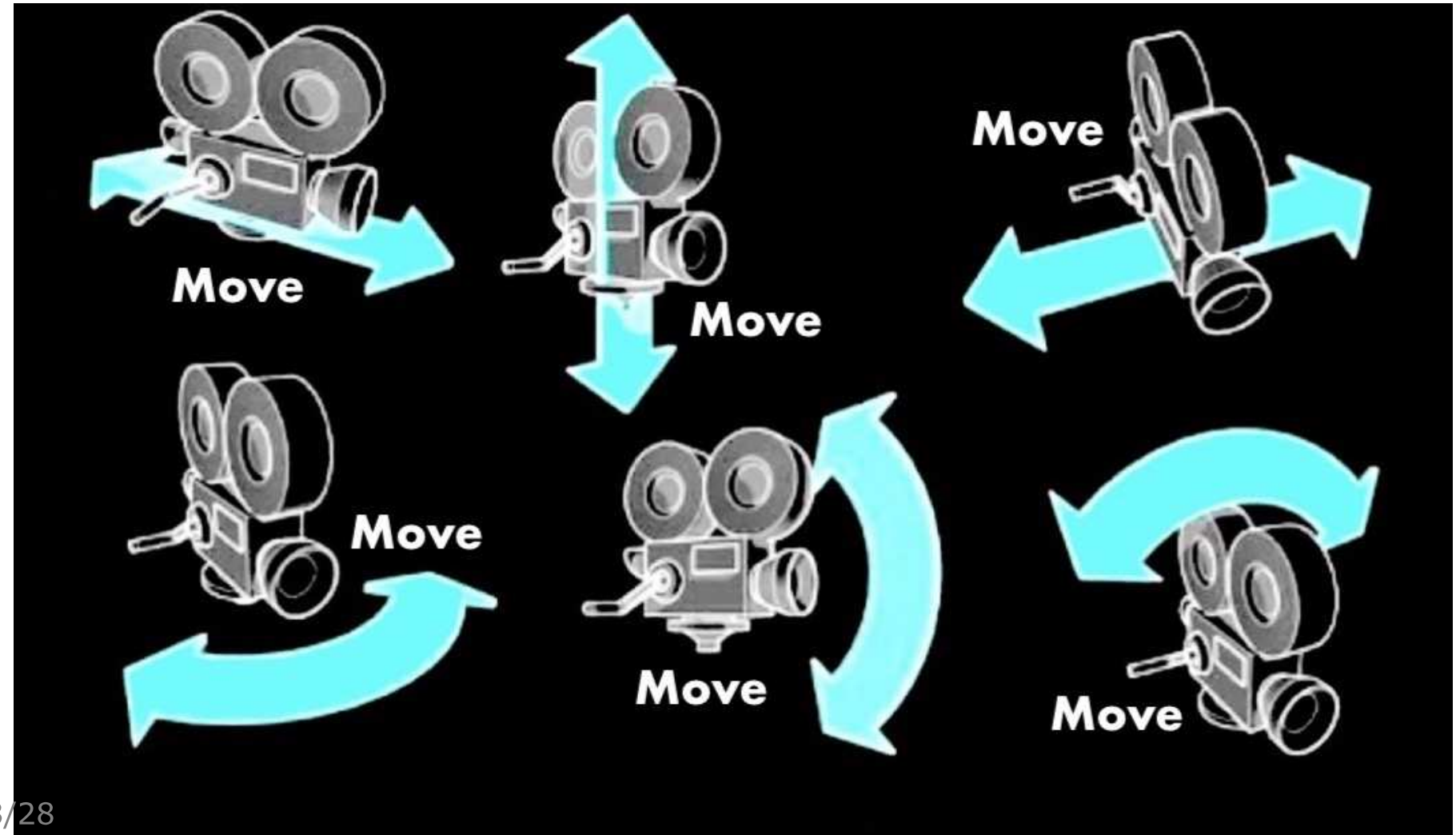
Cinematography Camera Movement



Client (Incorrect !) Camera Movement



Student (Nonspecific) Camera Movement



Optional 3D Texture Mapping

If you implemented 2D texture mapping last week

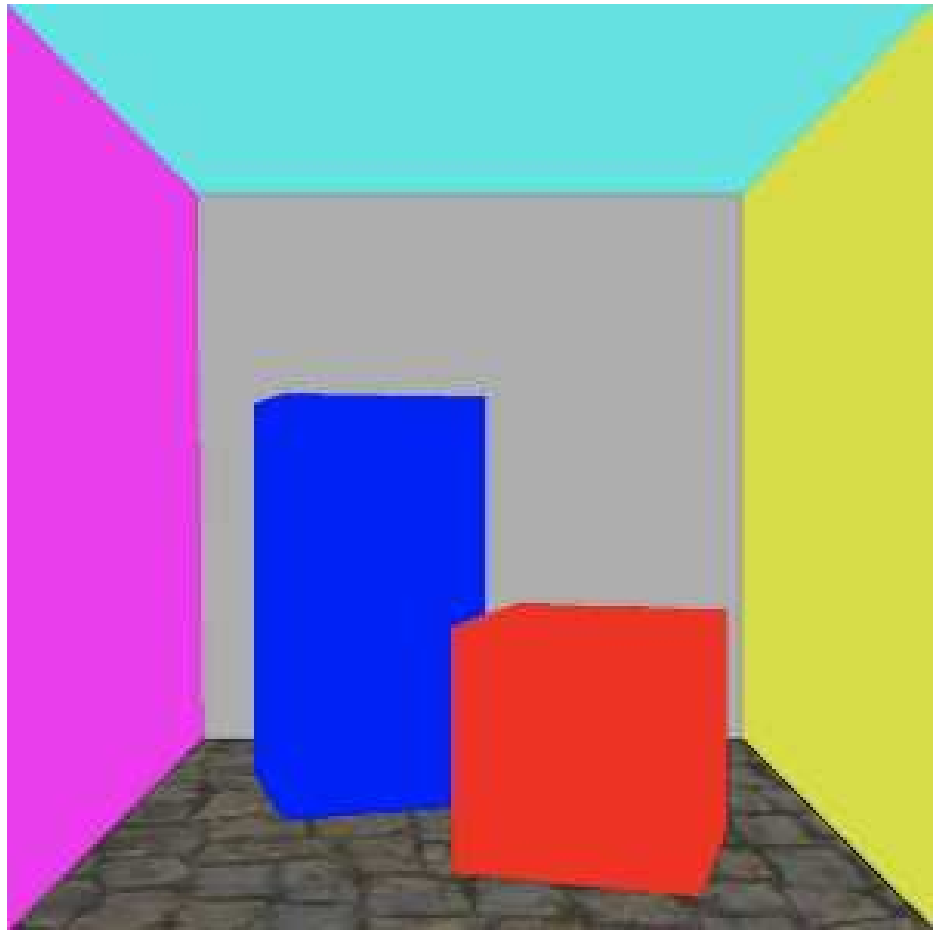
You can (optionally) use it in this week's worksheet

Updated OBJ files provide vertex-texture mappings

```
o floor
usemtl Cobbles
v -2.7470112 -2.7382329 2.806401
v 2.780989 -2.7382329 2.806401
v 2.780989 -2.742686 -2.785598
v -2.7150111 -2.742686 -2.785598
vt 0.1 0.1
vt 0.1 0.9
vt 0.9 0.9
vt 0.9 0.1
f 14/3 16/1 13/2
f 14/3 15/4 16/1
```

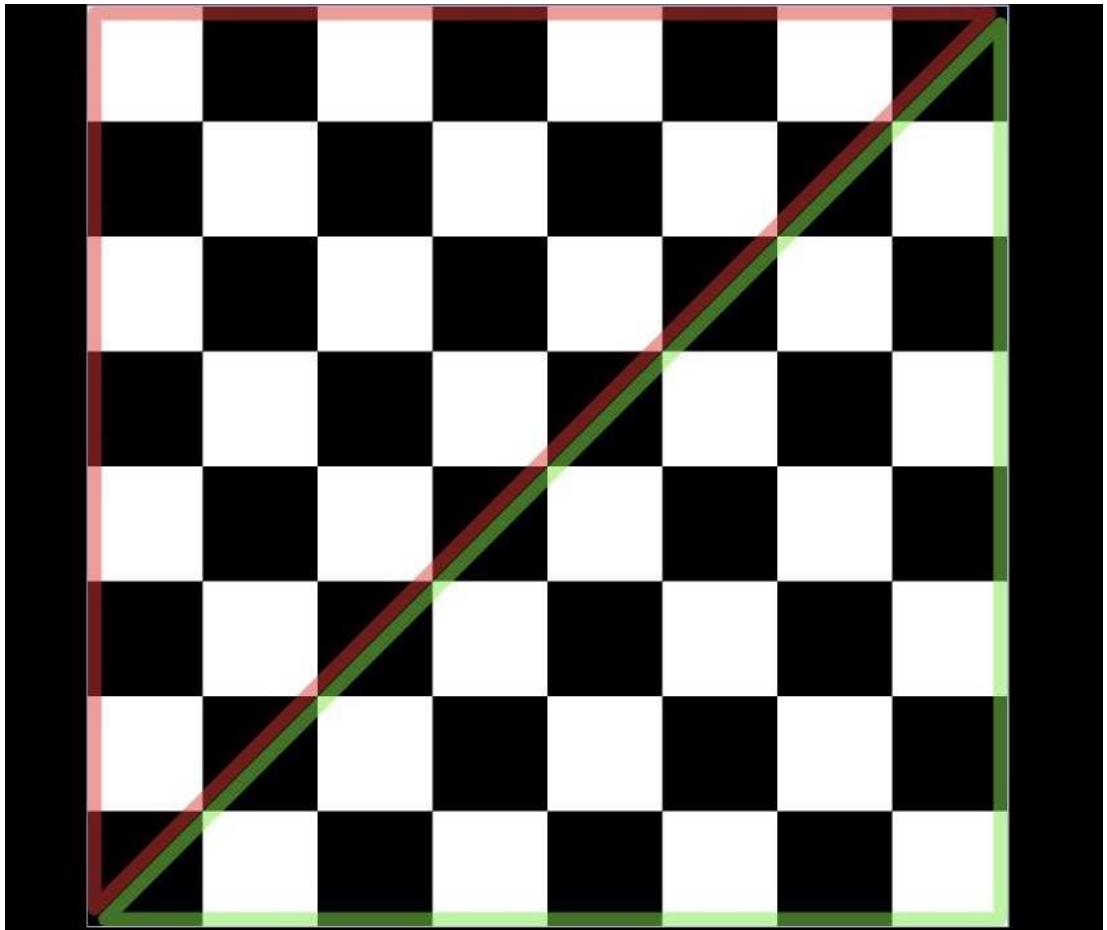
The *conceptual* objective is shown below

(but you won't get it this good !)



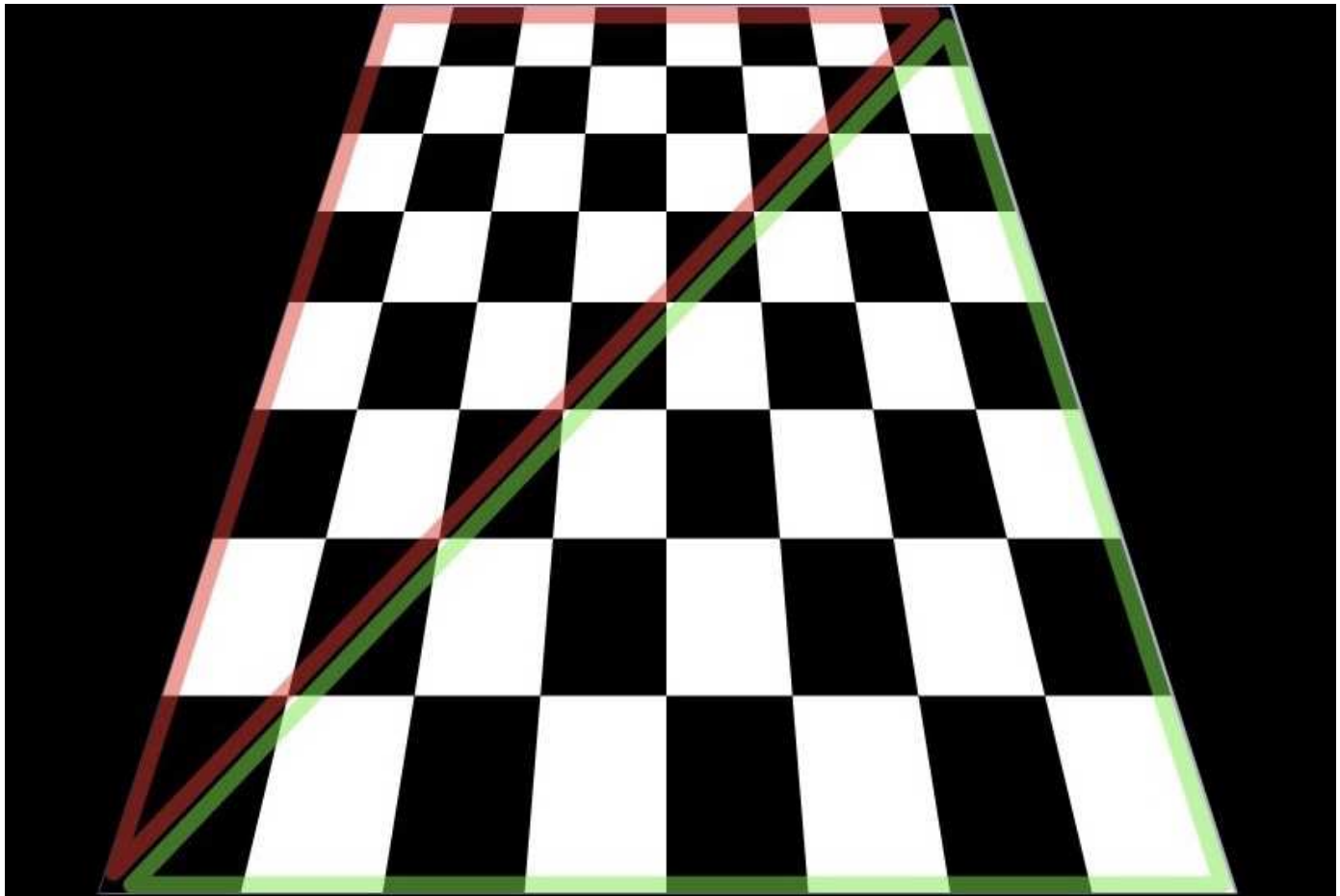
Consider the below "chessboard" texture

Two triangles used to render a flat square board



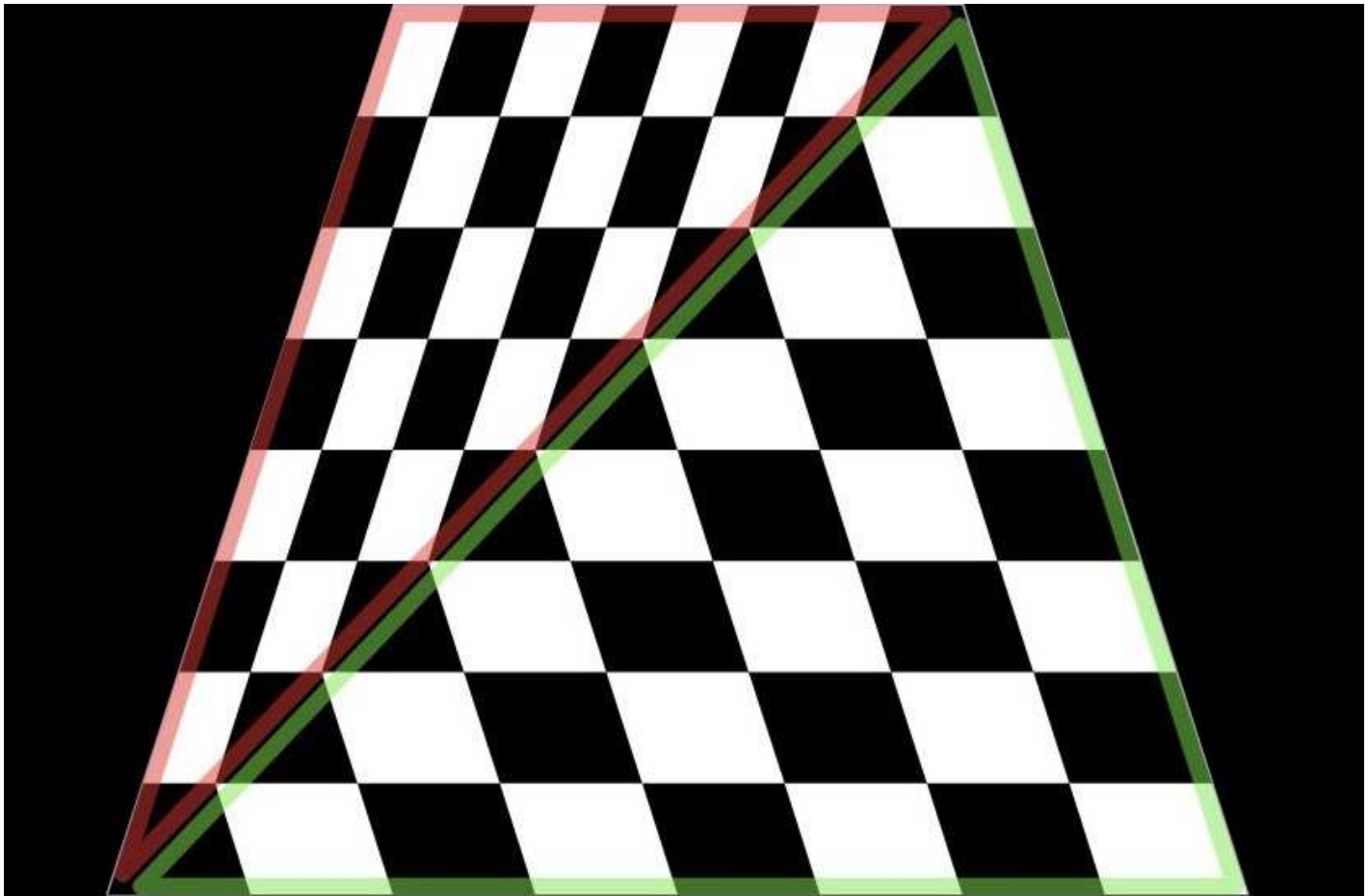
However, we are now working in 3D

Due to perspective, distant artefacts appear smaller



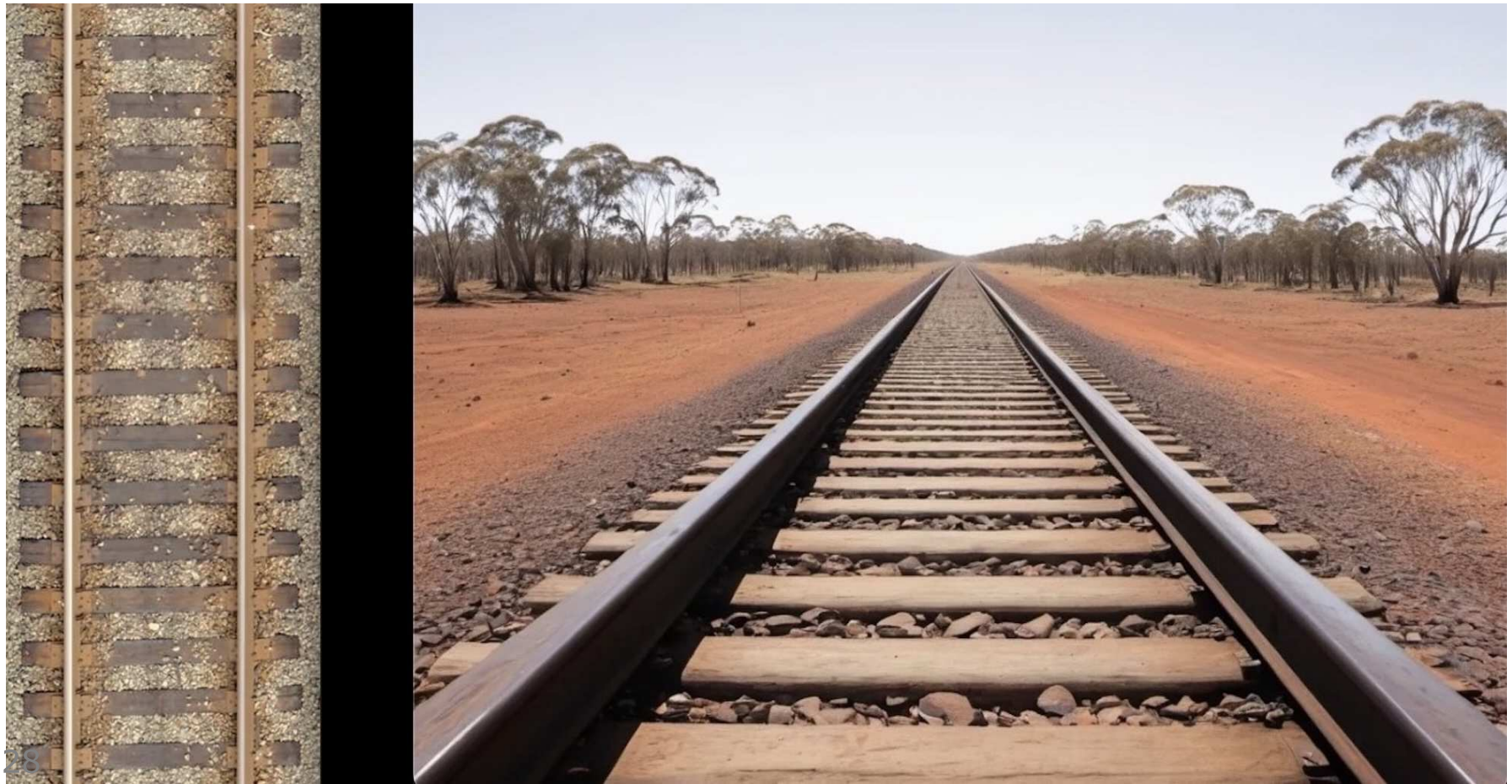
What YOUR texture mapper would produce

Currently it's just 2D (it doesn't consider Z depth)



Requires challenging non-linear sampling

Texture row step size increases incrementally !



But we aren't going to be doing that !

There is in fact a much easy way to texture in 3D

Which you might get the opportunity to explore later

Additional Guidance

Camera Orientation

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

Instead it makes reference to rear/back of camera

The reason for this is simple...
This representation just makes it
a lot easier to apply orientation
(when doing vertex projection)

We just multiply !

Overriden Operators

vec3 & mat3 override basic maths operators:

```
+  -  *  /
```

So we 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:

```
Position, 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 and Directions:

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

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 !

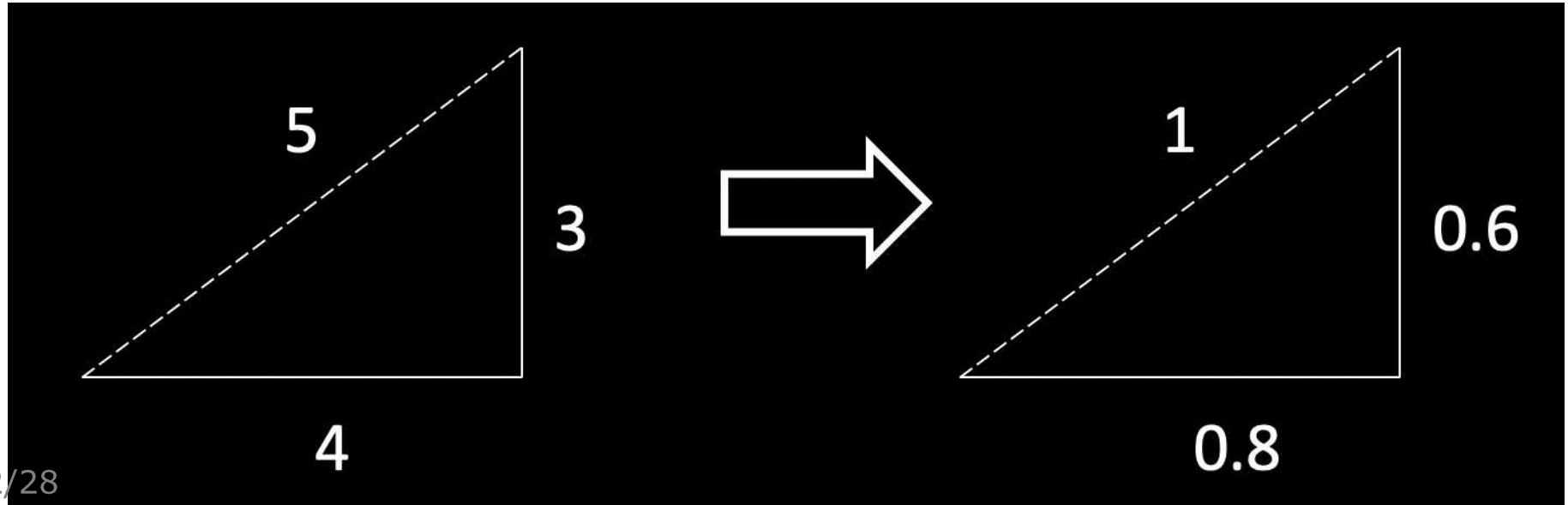
Normalisation

Important 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 ?

Good principle to work by:

If you NEED the distance - then DON'T normalise

Normalise at all other times !

The most popular approach:

If the render is broken

Randomly normalise a few vectors

(in the hope that it will fix things :o)

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
A3059
St Columb
Major
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