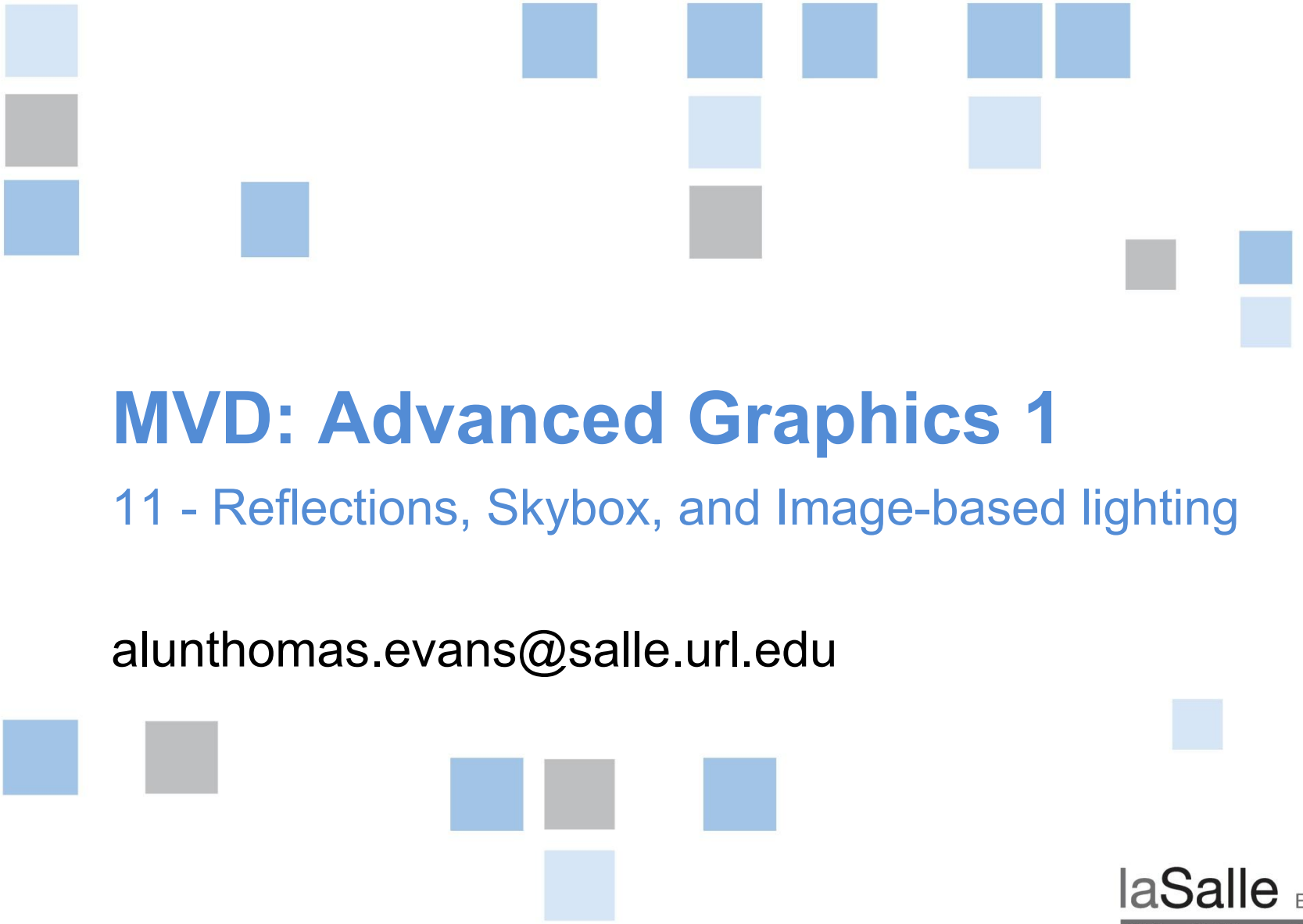


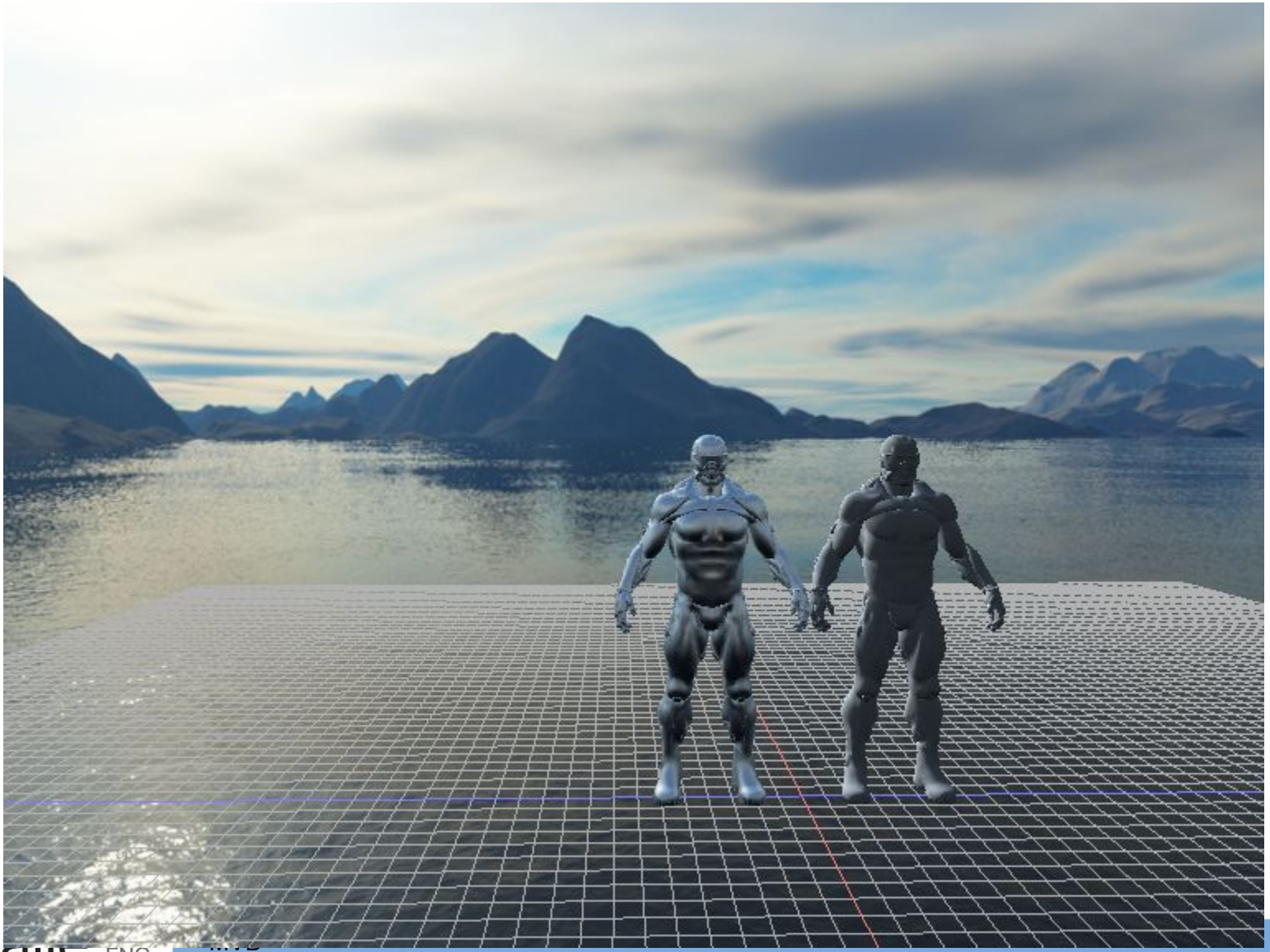
MVD: Advanced Graphics 1

11 - Reflections, Skybox, and Image-based lighting

alunthomas.evans@salle.url.edu

laSalle ENG
Universitat Ramon Llull

Reflections, Skybox and Image-based lighting

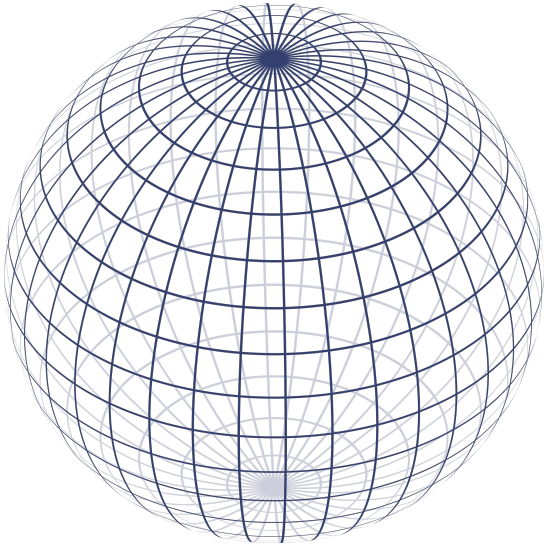


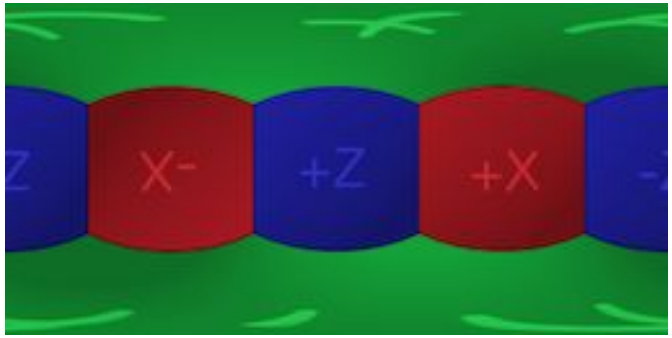
Code refactor

1st item on agenda: code refactor

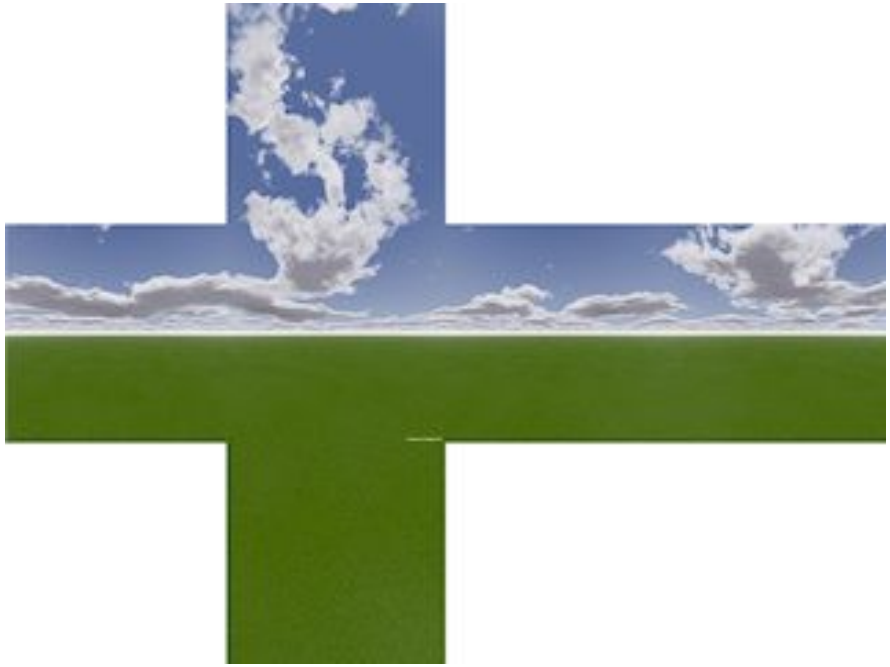
GraphicsUtilities

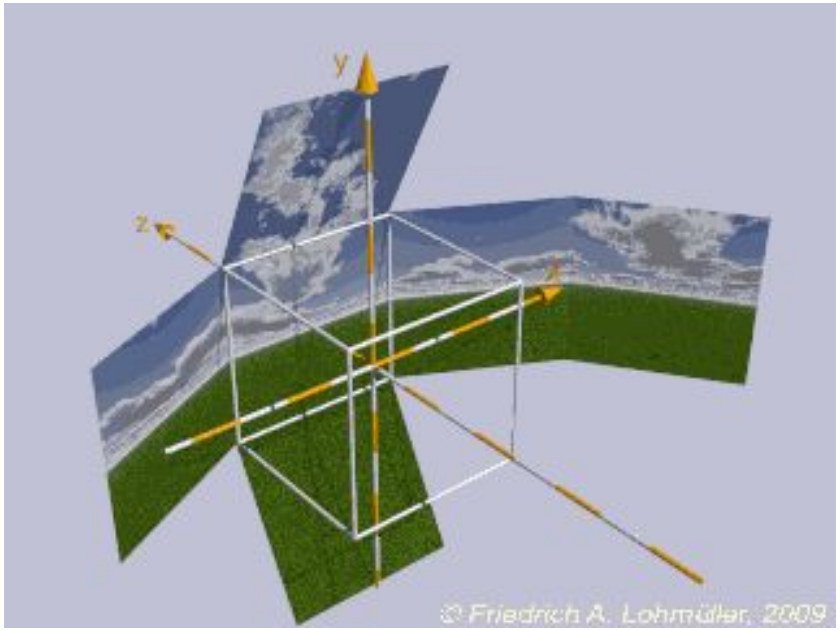
Skybox v1 - latitude/long sphere



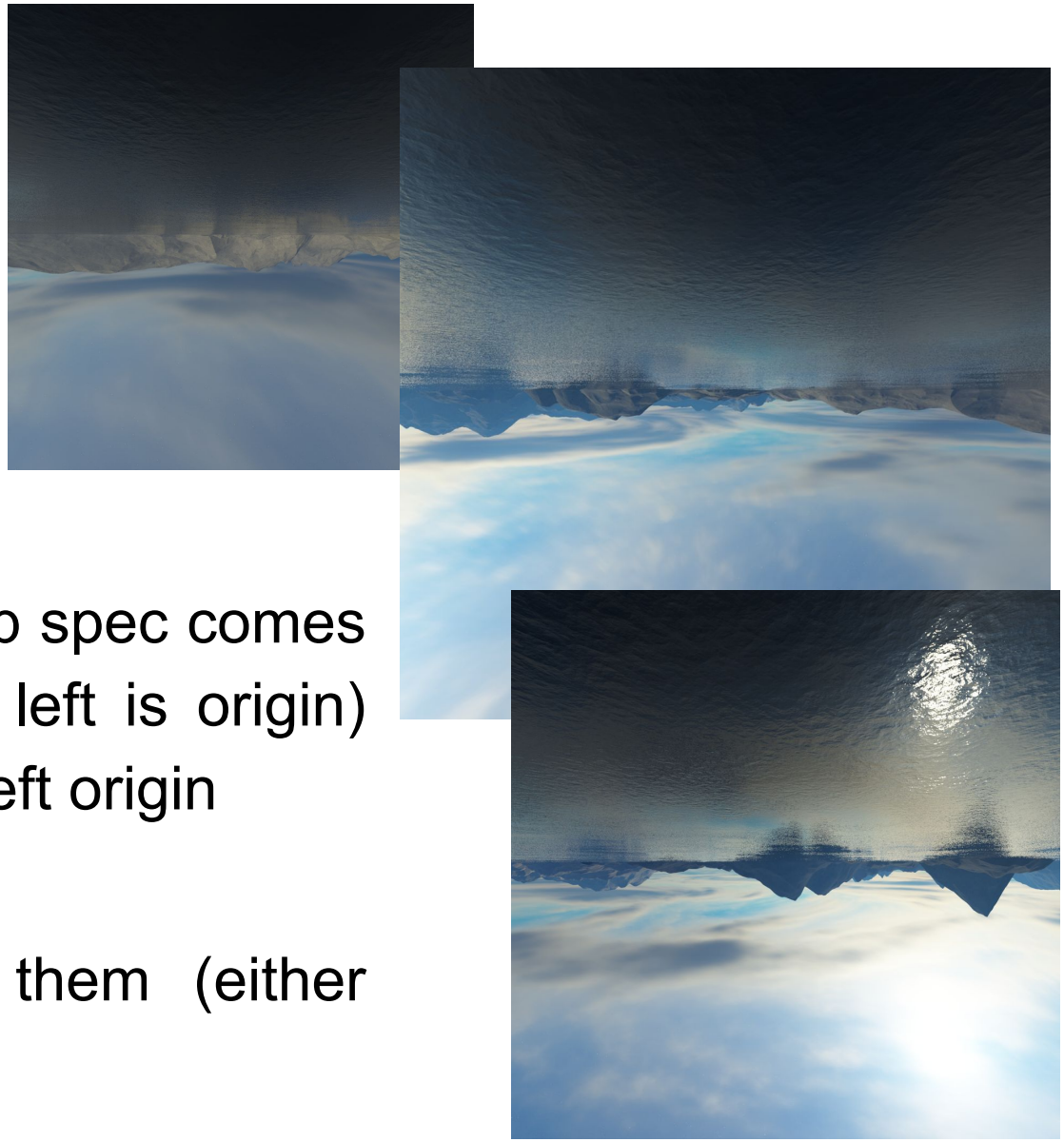


Skybox v2: Cubemap





Skybox v2.5



6 separate images

They’re upside down!?

Yes because Cubemap spec comes from Renderman (top left is origin) and opengl is bottom-left origin

So we need to flip them (either manually or in code)

To create a cubemapped skybox you need

1) Geometry

   (simple box geometry)

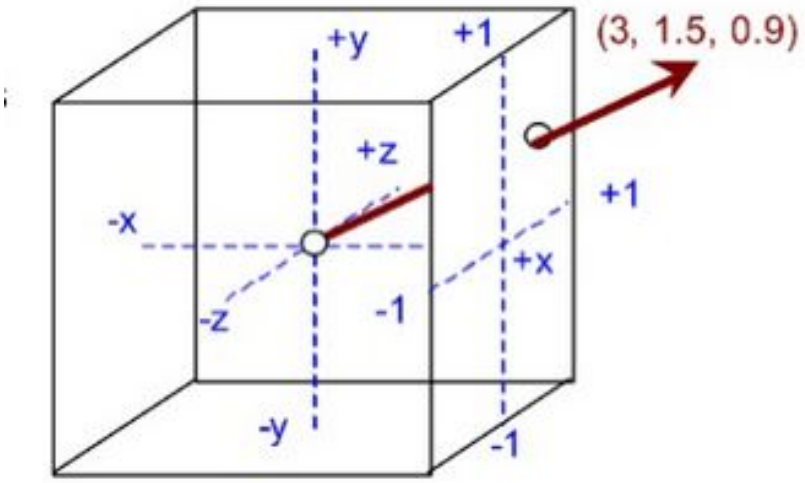
2) Texture(s)

   (either 6 individual, or cubemap format

3) Shader

Geometry

Just a regular cube



The interesting thing is you don't need regular 2D texture coordinates - you use the vertex positions as texture coords

When a cube is centered on the origin (0,0,0) each of its **position vectors** is also a direction vector from the origin.

This direction vector is exactly what we need to get the corresponding texture value.

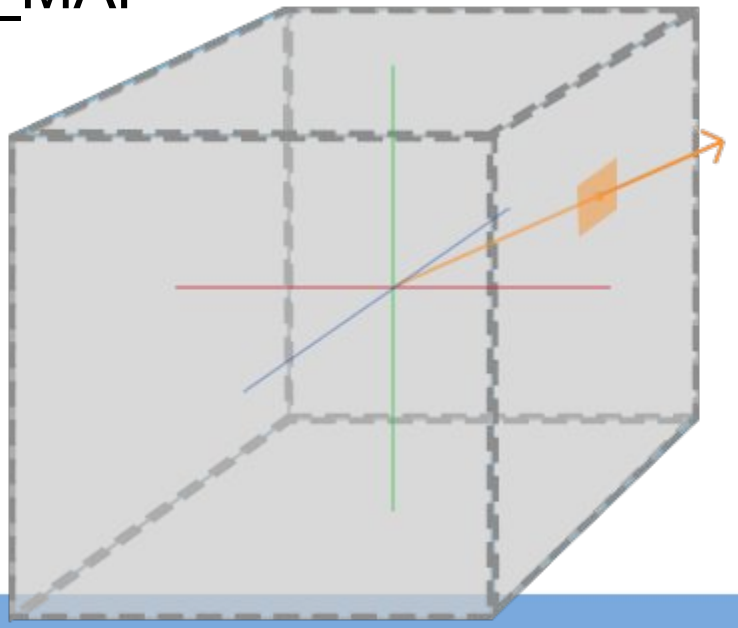
Loading cubemap textures

OpenGL supports a special type of texture and sampler.

The texture is GL_TEXTURE_CUBE_MAP

The sampler is called *samplerCube*

Both assume a cube geometry



Loading cubemap textures

Very similar to loading regular textures, but you have to specify data pointers for all 6 directions:

```
//Define all 6 faces
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X, 0, GL_RGB, width, height, 0, GL_BGR, GL_UNSIGNED_BYTE, tgainfo0->data);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X, 0, GL_RGB, width, height, 0, GL_BGR, GL_UNSIGNED_BYTE, tgainfo1->data);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y, 0, GL_RGB, width, height, 0, GL_BGR, GL_UNSIGNED_BYTE, tgainfo2->data);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, 0, GL_RGB, width, height, 0, GL_BGR, GL_UNSIGNED_BYTE, tgainfo3->data);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z, 0, GL_RGB, width, height, 0, GL_BGR, GL_UNSIGNED_BYTE, tgainfo4->data);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, 0, GL_RGB, width, height, 0, GL_BGR, GL_UNSIGNED_BYTE, tgainfo5->data);

glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_BASE_LEVEL, 0);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAX_LEVEL, 10);
glGenerateMipmap(GL_TEXTURE_CUBE_MAP);
```

In our engine

```
std::vector<std::string> cube_faces{
"data/assets/skybox/right.tga",
"data/assets/skybox/left.tga",
"data/assets/skybox/top.tga",
"data/assets/skybox/bottom.tga",
"data/assets/skybox/front.tga",
"data/assets/skybox/back.tga"
};
GLuint cubemap_texture = Parsers::parseCubemap(cube_faces);
```

The cubemap shader

Pretty standard *except*

texture varying is a **vec3, not a vec2**

```
out vec3 v_tex; //note: vec3!
uniform mat4 u_vp;

void main(){
    //v_tex is a vec3, not a vec2
    v_tex = a_vertex;

    //calculate position
    vec4 pos = u_vp * vec4(a_vertex, 1.0);
    gl_Position = pos;
}
```

Rendering the cubemap

Where's the model matrix?

Where's the cubemap?

How do we ensure that we draw it behind everything else?

Rendering the cubemap

Where's the model matrix?

Set translation component of View Matrix (*only for cubemap*) to 0...

Where's the cubemap?

….so it is always centered on camera

How do we ensure that we draw it behind everything else?

Disable depth-test, and draw it *before* everything

Task: cubemap shader

1) Vertex shader:
   a) create out varying (vec3) to pass texture coords to fragment shader, and fill it with raw vertex coords
   b) create u_vp uniform (mat4)
   c) glPosition should be u_vp * vertex coords
2) Fragment shader
   a) create in varying (vec3) for texture coords
   b) create uniform (samplerCube) for skybox
   c) write fragColor by sampling with texture()

Task - complete renderEnvironment_()

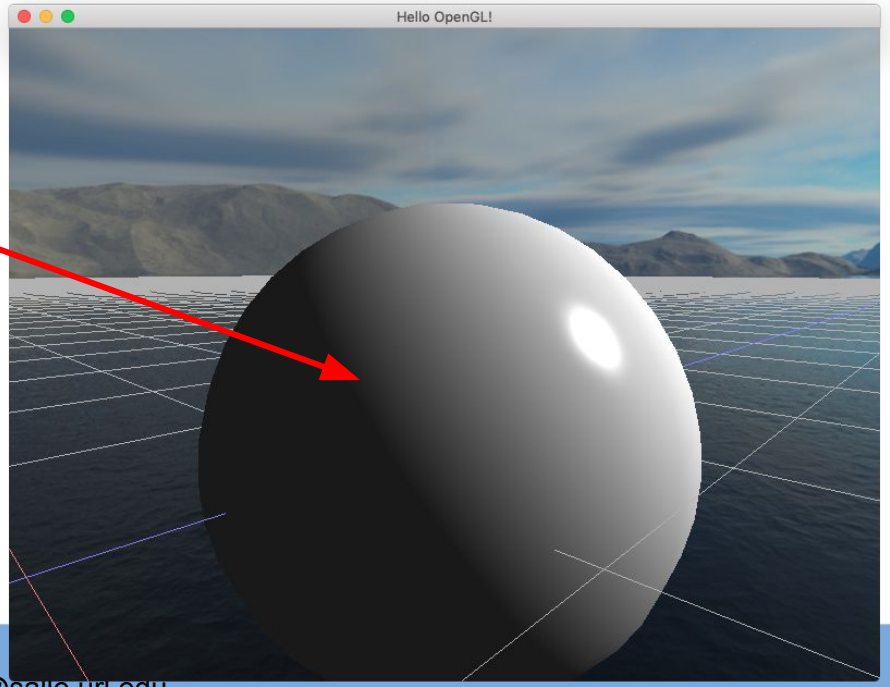
1) Uncomment code in game.cpp
2) Create a VP matrix with translation element of view matrix set to zero
3) set VP uniform in shader
4) bind *environment_texture* to GL_TEXTURE0 using GL_TEXTURE_CUBE_MAP
5) use glDepthMask() and glCullFace to disable depth test and set culling
6) bind geom.vao, and draw
7) rset depthMask and face culling
8) call renderEnvironment_() from update

An optimisation

Rendering the skybox first isn't very optimal.

Why? Because if our scene is filled with other stuff, we are doing the pixel calculations for the skybox, and then overwriting them

all this bit behind sphere is wasted



An optimisation

So want to render skybox last, but depth test will draw it on top of everything.

How do we trick the depth test into thinking that the skybox is ‘really far away’?

Set depth of skybox to 1

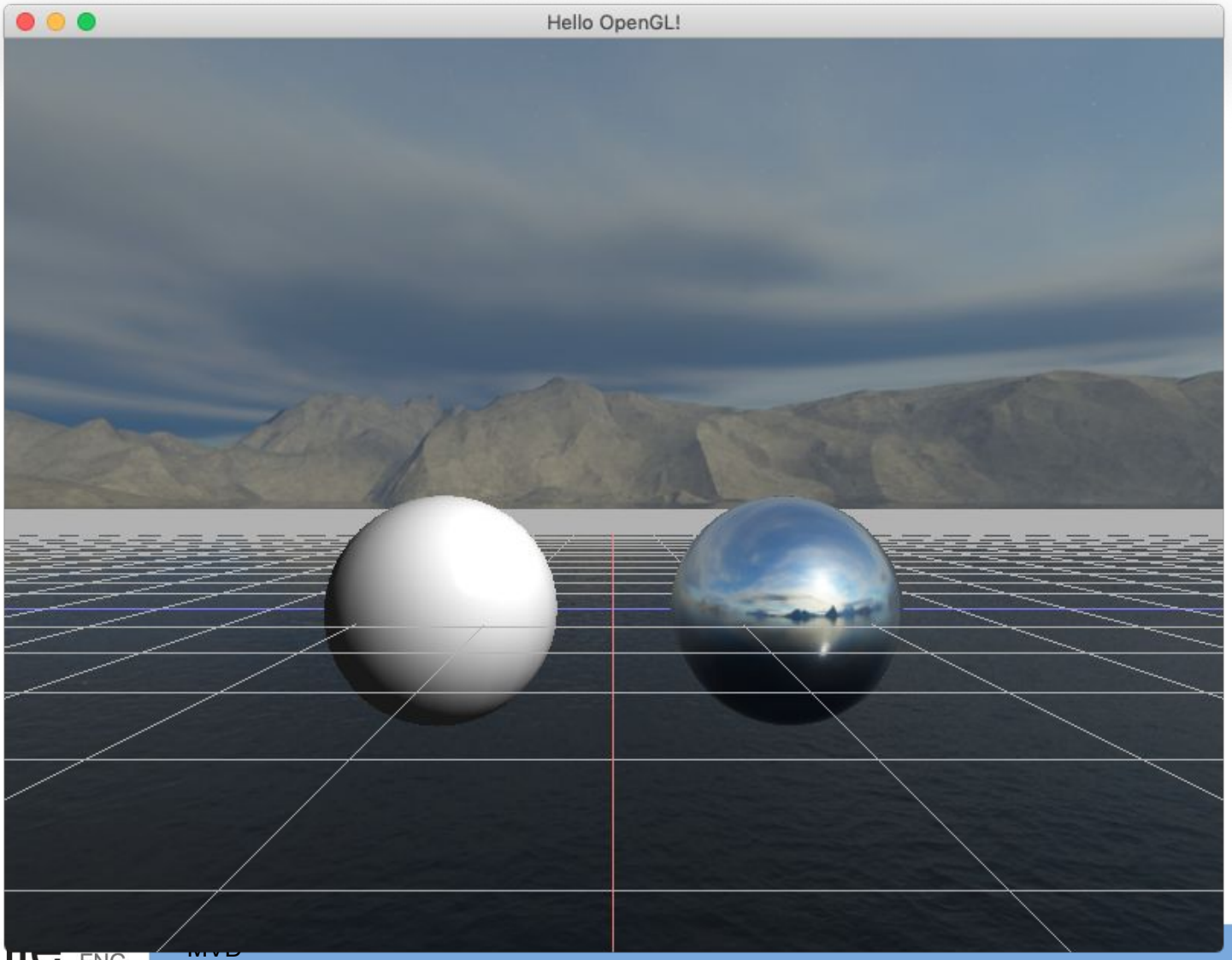
Use homogenous coords to set depth of gl_Position to w component.

Homogenous coords divide by *w* to get position in NDC

So now depth buffer thinks that all skybox fragments have z value of 1!

```
//calculate position
vec4 pos = u_vp * vec4(a_vertex, 1.0);
//gl_Position = pos;
//optimisation
gl_Position = pos.xyww;
```

Reflections



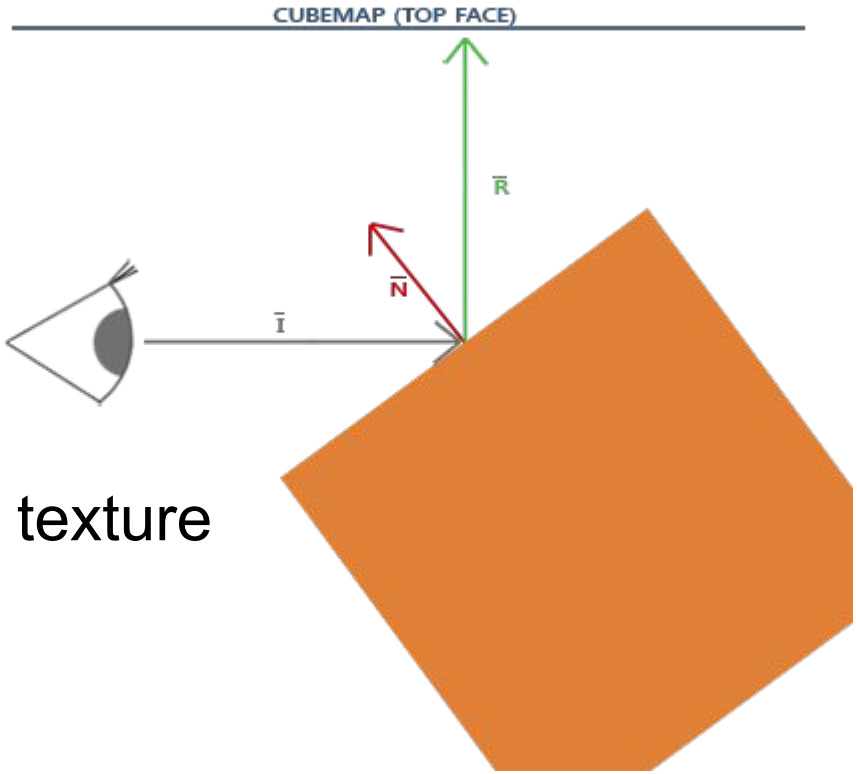
Reflection

Calculate reflection vector from surface

Sample cubemap texture

A form of ray tracing

Slightly more costly than regular texture sampling, but not very much



Add cubemap id to material

Need to add cubemap id to material. Will send this to reflection shader

```
struct Material {
    std::string name;
    GLuint shader_id;
    lm::vec3 ambient;
    lm::vec3 diffuse;
    lm::vec3 specular;
    float specular_gloss;

    GLuint diffuse_map;
    GLuint cube_map;
```

set texture uniform when drawing

New abstracted function in Shader.cpp

```
shader_->setTextureCube(U_SKYBOX, mat.cube_map, 1);
```

Task - write reflection shader

1) Calculate incident vector (from camera to vertex)
2) Calculate reflection vector
3) Sample skybox
4) Write to fragColor

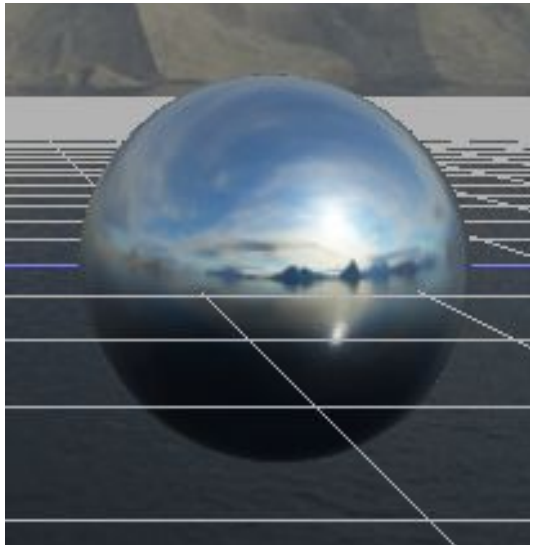
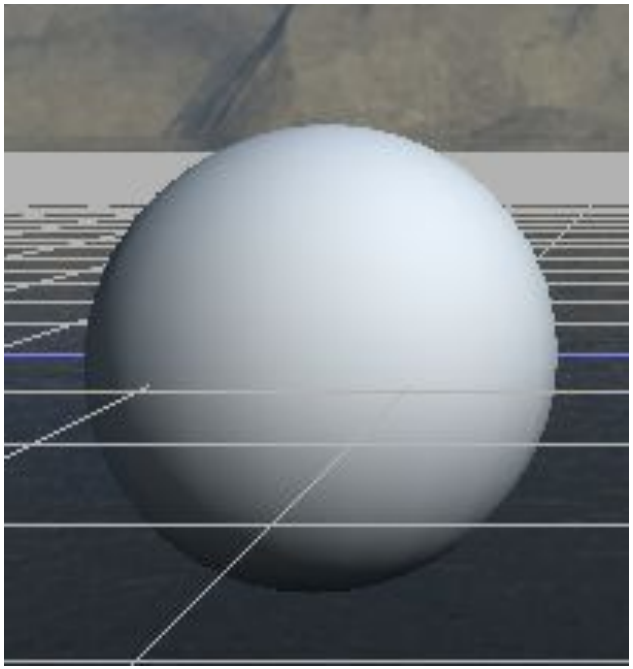
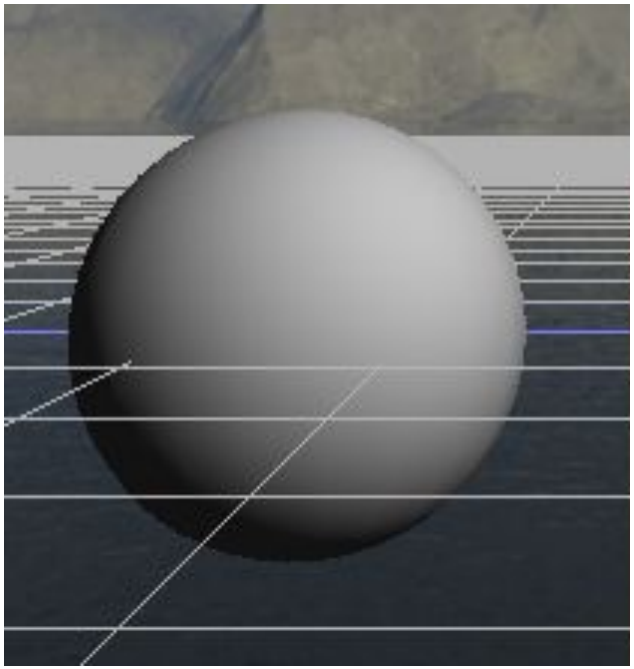


Image based lighting

Use environment to provide ambient light color

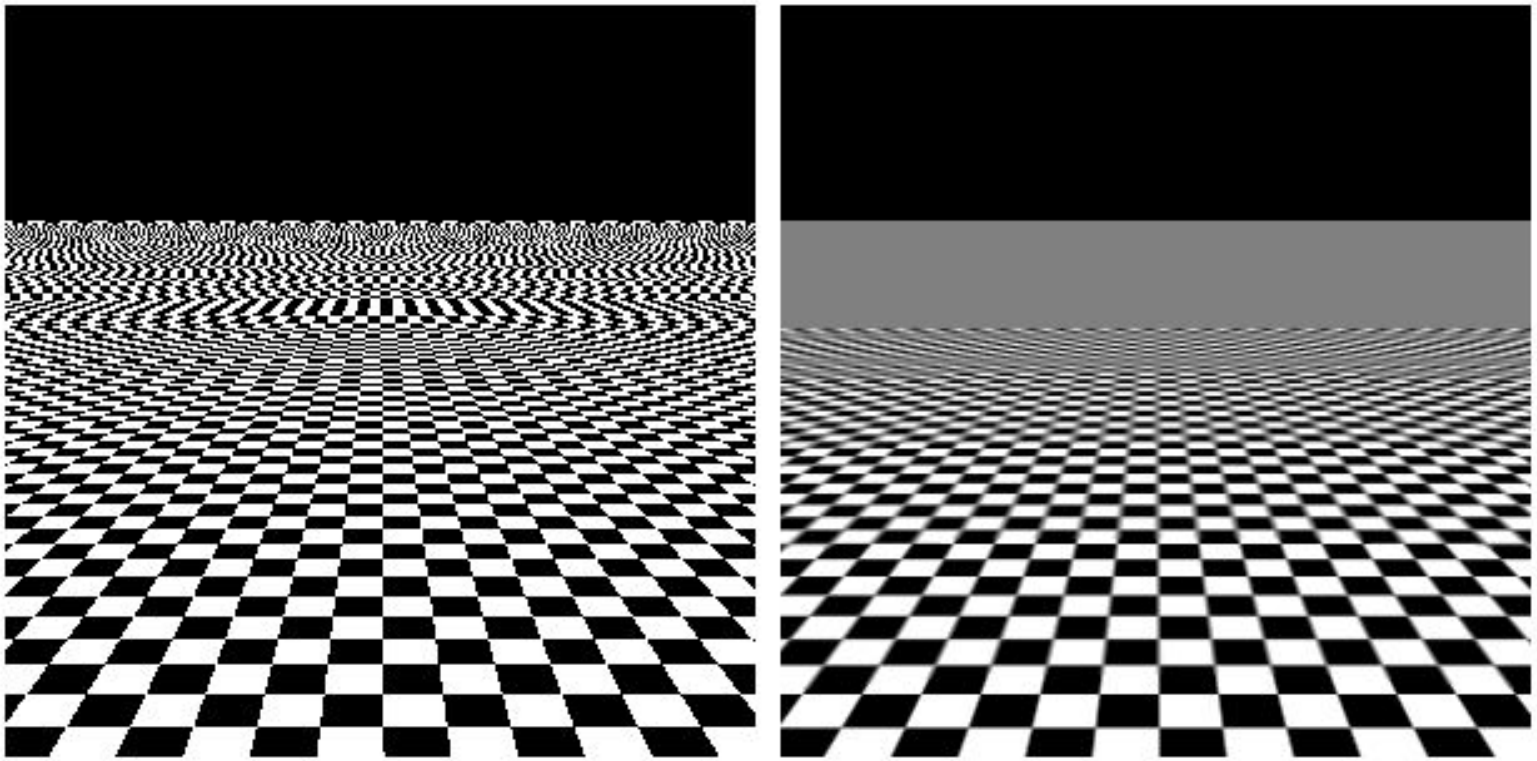




Mipmaps

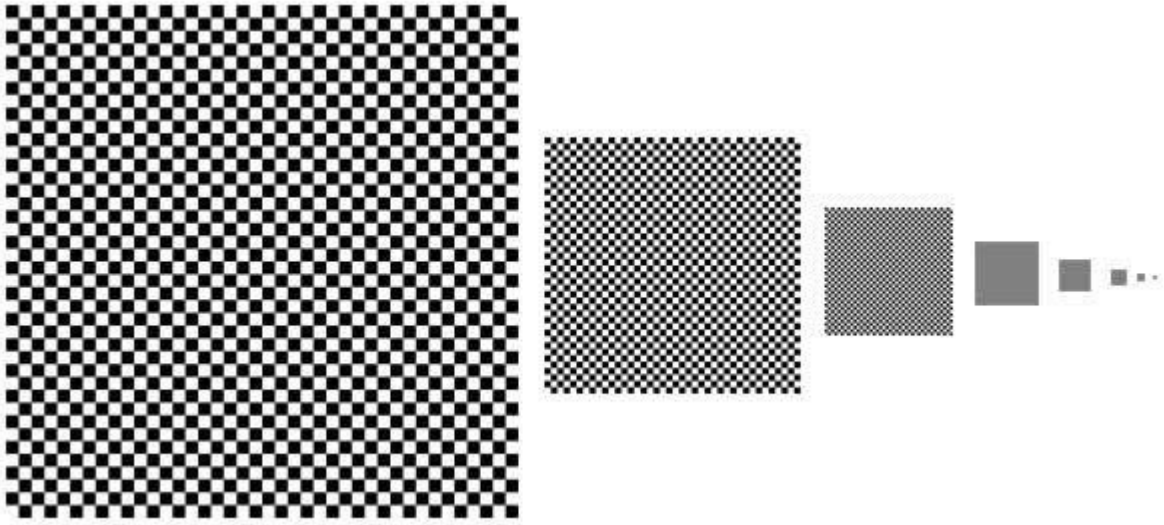
Mipmapping is the use of lower level of detail textures to reduce rendering artifacts to due aliasing of textures, caused by perspective projection

"*Humans hate aliasing*"



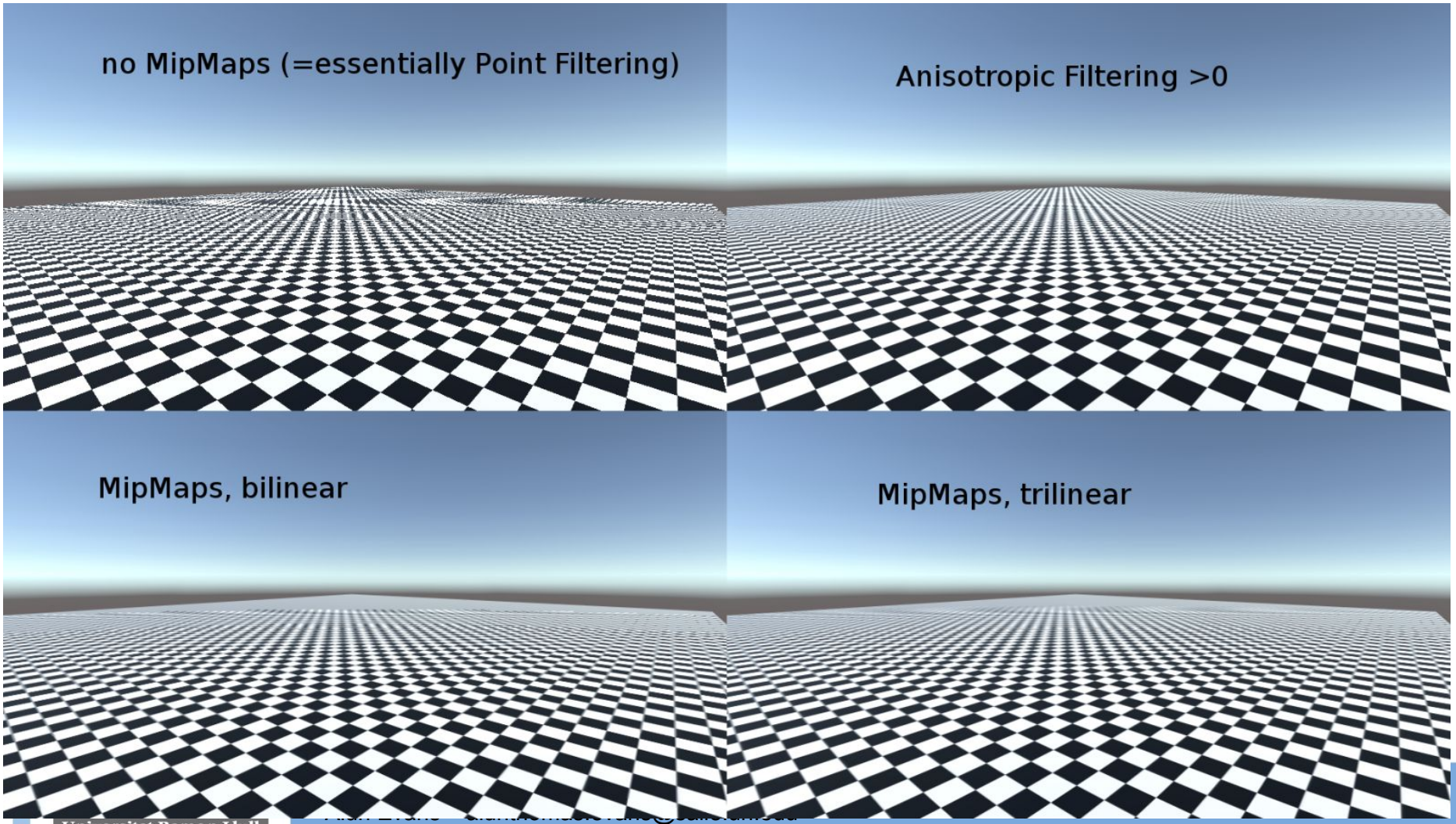
Mipmaps

We sample lower LoD detail textures at distances further from the camera



There are various filtering methods, linear, bilinear, trilinear, anisotopric

Mipmap filtering



The GPU creates our Mipmaps for us

```
//we want to use mipmaps
glGenerateMipmap(GL_TEXTURE_2D);
```

or for cube maps

```
glGenerateMipmap(GL_TEXTURE_CUBE_MAP);
```

we call this function when we create the texture object

We can hard code the number of mipmap levels:

```
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_BASE_LEVEL, 0);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAX_LEVEL, 10);
```

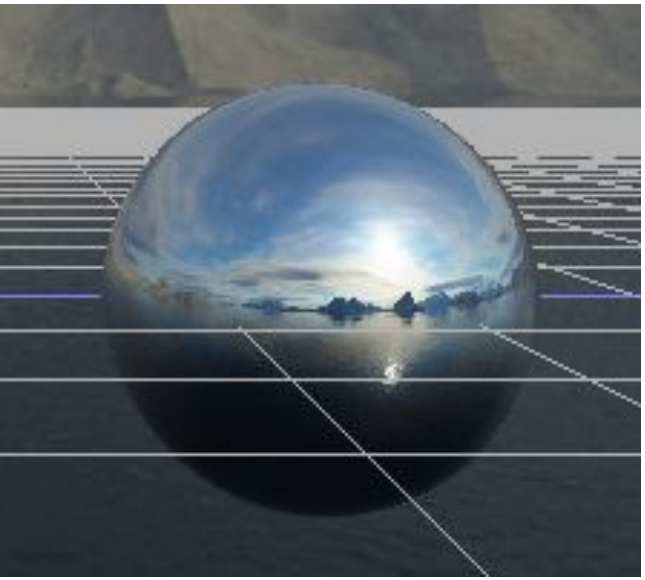
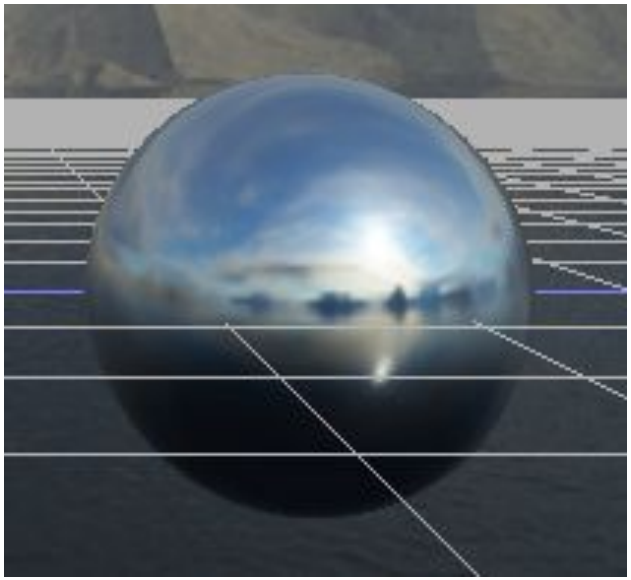
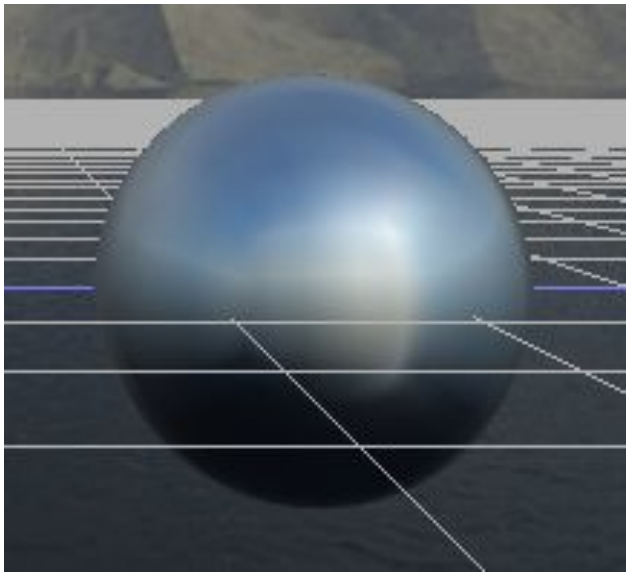
Mipmapping - other uses

The primary use of mipmapping is to reduce texture aliasing and it all done automatically on the GPU.

However, we can access the mipmap level of details in the shader, using the textureLod function

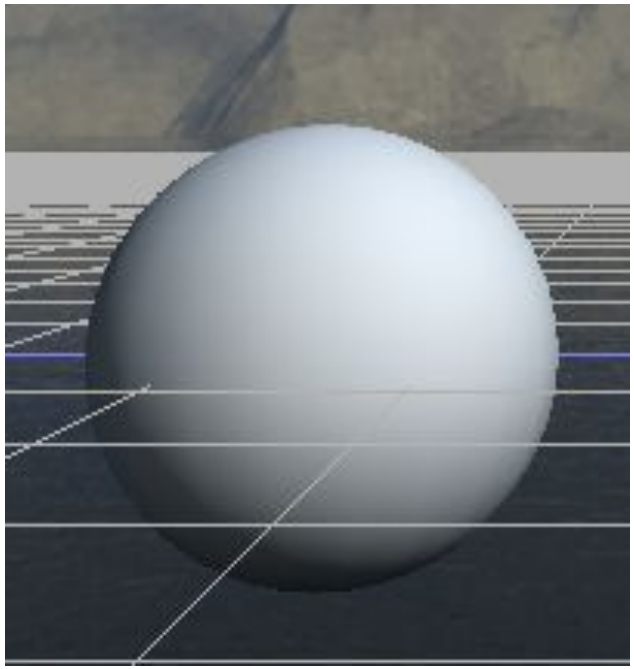
```
vec3 reflect_color = textureLod(u_skybox, R, 5.0).rgb;
```

LoD 3 - 5 - 8

LoD 10

For our texture, at LoD we get an idea of the colour, but no reflection:



Task: Modify phong sphere/shader

1) Create a copy of phong fragment shader and rename
2) In new shader:
   a) add skybox uniform
   b) multiply ambient color by textureLoD of 10.0
3) In C++
   a) set ambient and diffuse color of sphere material to (0.7,0.7,0.7)
   b) modify setMaterialUniforms to set skybox texture