F20GA

Graphics and Animation
MACS, Heriot-Watt University

Coursework Report (November 2020)

Benjamin CJ Milne
H00267054
<bm56@hw.ac.uk>

Introduction

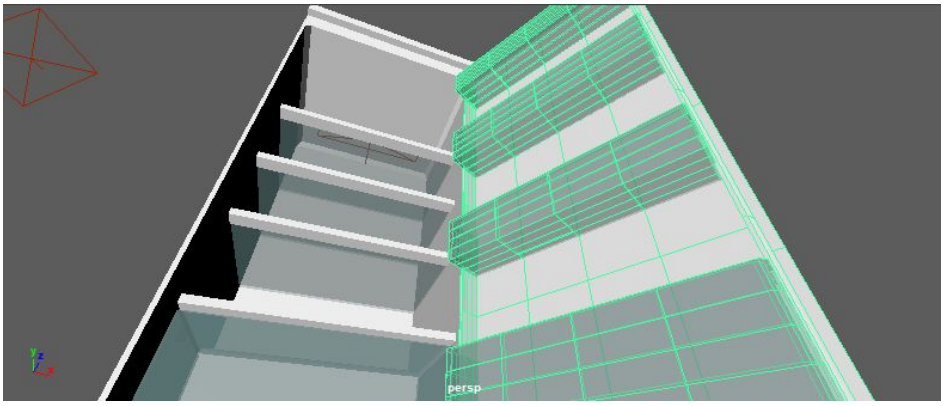
I often lack creative inspiration, so I decided to model something close to my heart: a fridge. I felt this would have adequate moving pieces and simple enough geometry to model in the timeframe provided, while still allowing for the demonstration of more interesting materials and renders such as with the transparency of the shelves.

HQ Render

Modelling

I used Maya to create the model and Arnold to create the HQ render. I come from using Autodesk products such as Inventor, so it was interesting not using strict measurements when creating the object. I used a lot of **Boolean Difference** operations to hollow out various pieces of the fridge, such as the main body, and discovered that you need to **Freeze Transformations** to essentially 'apply' a scale factor to an object in order for rotation to apply correctly.

I have also used **grouping** to allow me to move a number of objects at once. I moved the **pivot point** for the door grouping to around where the hinge would be, to allow me to rotate it realistically:

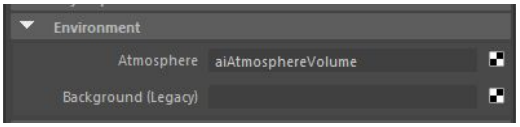


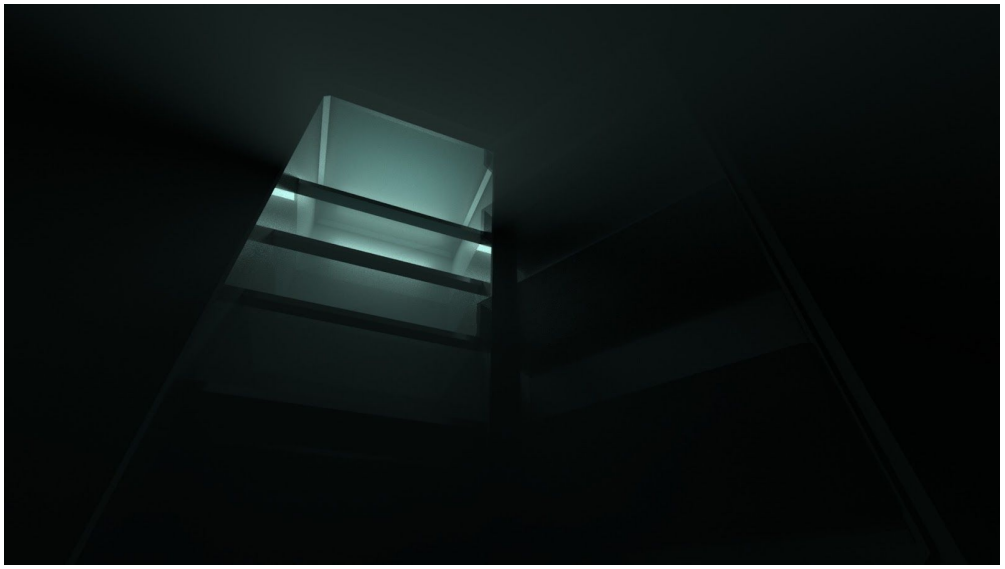
I discovered the **Arnold Renderview** window which allowed me to have a (low-quality) preview of the Arnold render open, changing in "real-time" as I edited the model and materials in the main window.

The body and door material is a **Phong** type which adds an element of reflection, while the plastic shelves are a 50% transparent **Lambert**.

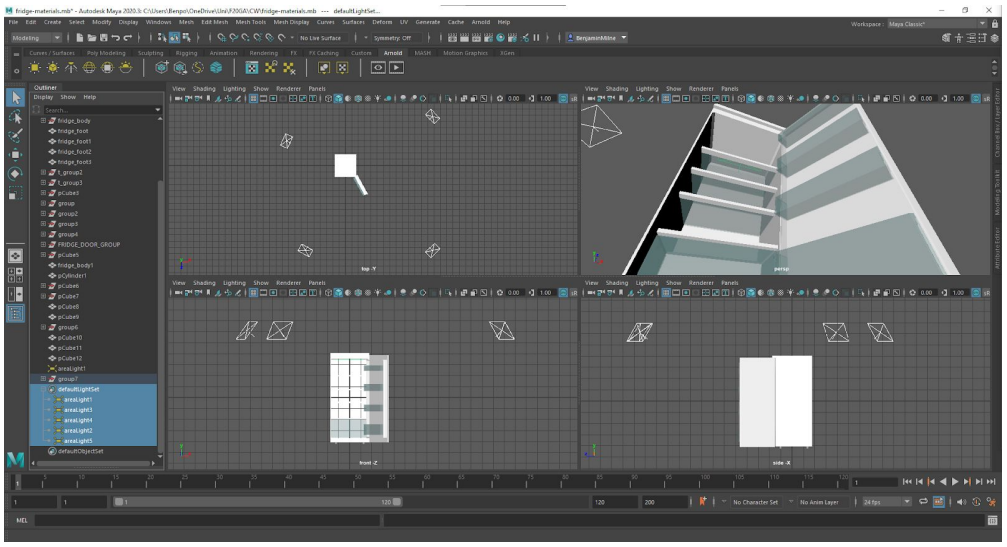
Lighting

I explored using Arnold's **atmosphereVolume** to create a dramatic atmosphere. In my head, I was thinking either "fridge from a horror movie" or "television advert for chicken strips":





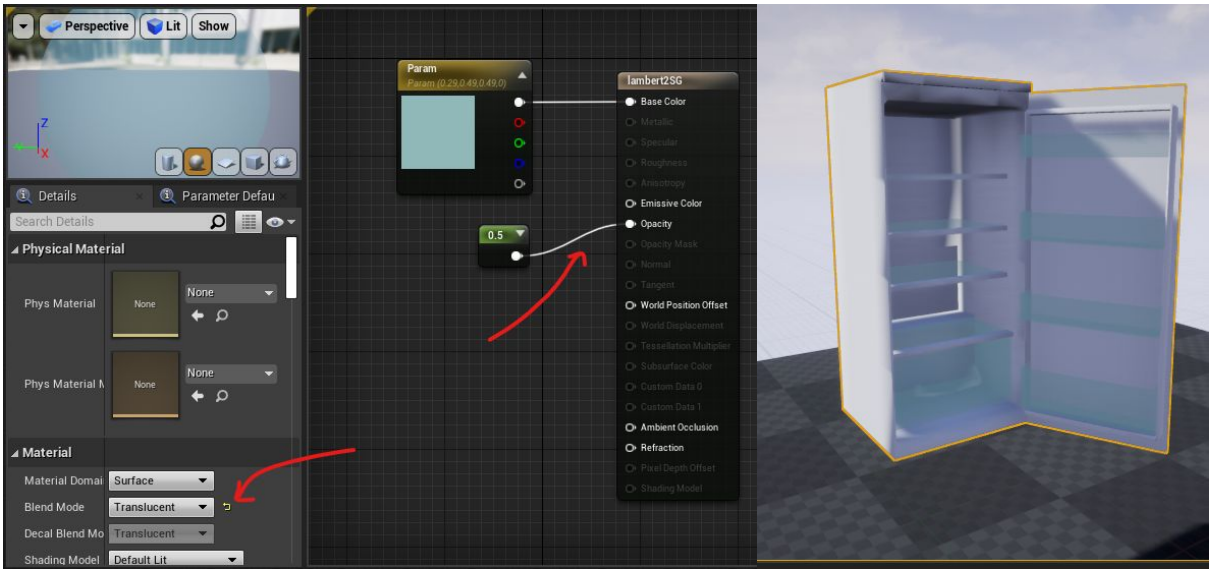
To create this image, I used four **areaLights** facing diagonally in from above on low intensities so that the edges of the model were still visible in the final render, and a fifth **areaLight** on the inside of the fridge itself, around where a real internal fridge light would be:



Animation

Importing the models

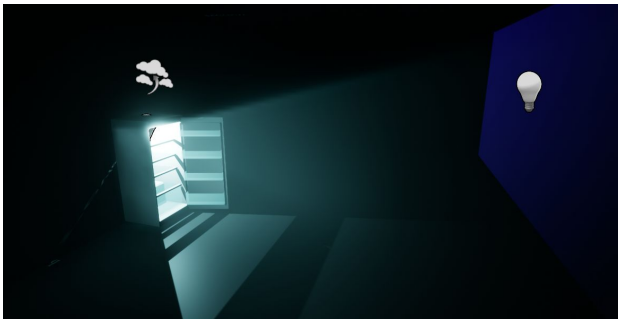
I exported the body components and door components from Maya as two OBJ files before importing them into a blank Unreal project. I had to scale up the objects by 20x, and edit the transparent plastic material to re-introduce transparency by changing the Blend Mode to Translucent and providing an opacity value:

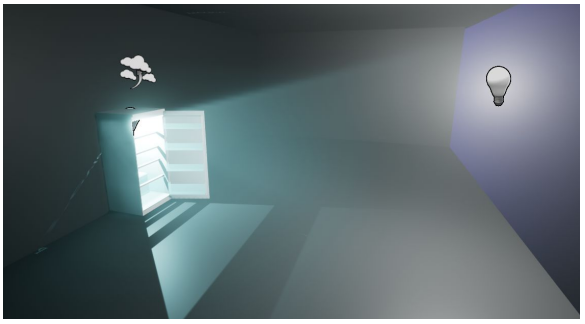


I exported the two pieces separately so that I could move the door independently of the body in the animation. This might be possible with skeletons, too, but I did not manage to work that out (it would always deform the object).

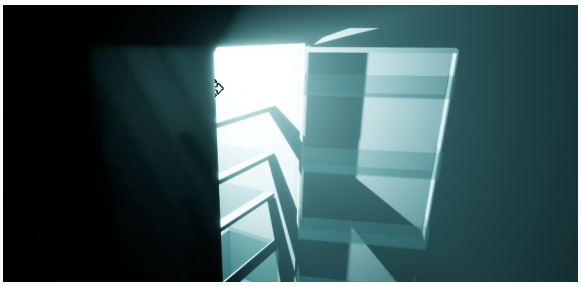
Lighting

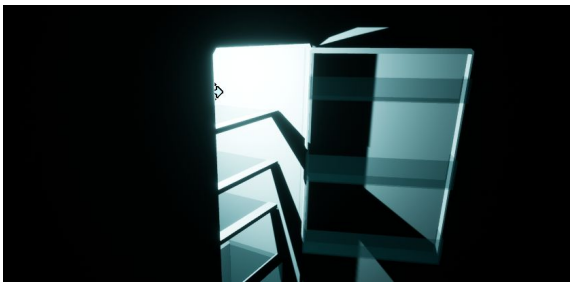
I grouped and converted the imported pieces into two **Static Mesh** objects which allowed me to permanently change the pivot point for the door to around where the hinge would be on a real fridge, making it easier to animate, and added an **ExponentialHeightFog** object to recreate the fog rendered in Maya. I removed the starting light and added a **Point Light** to the inside of the fridge, added walls and created a basic grey texture for them.





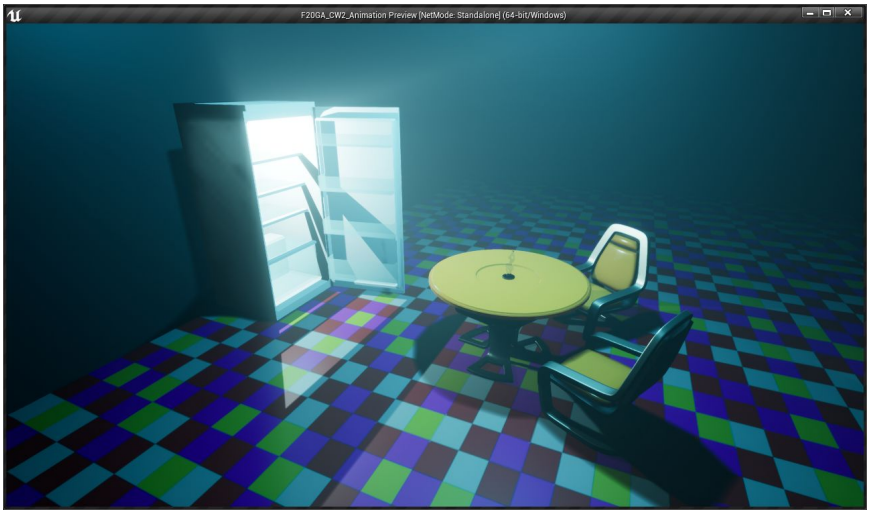
Unfortunately, a bi-product of the fog is that it makes the light appear as though it is “glowing” through the body of the fridge:



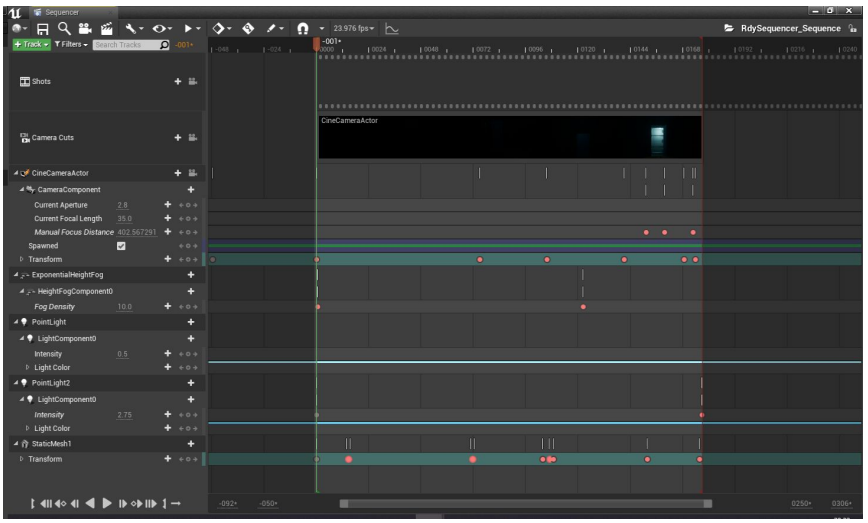


Fog enabled versus **Fog disabled**

Finally, I added additional props from Unreal’s Starter Content to add a little bit of context to the scene, before using **Sequencer** to animate the final video.



I fiddled with a lot of Actor parameters in Sequencer to get the final video looking correct, including increasing and decreasing light intensity, and decreasing fog density from an initially high value in an attempt at displaying the volumetric fog while the camera exposure was increasing.



Interactive Program

Importing the models

When importing the OBJ models into the OpenGL template, I discovered our OpenGL setup does not render quads correctly, so I went back into Maya and selected **Mesh > Triangulate** before exporting the OBJ objects again. *(I think I could have changed the .Draw() method also).*

I converted the **model**, **modelPosition** and **modelRotation** variables into **arrays**, to store multiple meshes:

```
const int     noModels = 15;
Mesh          model[noModels];          // Array of loaded models
glm::vec3     modelPosition[noModels];  // Array of model positions
glm::vec3     modelRotation[noModels];  // array of model rotations
```

This allowed me to import and store multiple models, which I crudely imported and manually aligned. Instead of instancing, I duplicated model imports for the fridge shelves, which is memory-inefficient; however I preferred the simpler rendering structure to the marginal performance increase:

```
// Load main object model and shaders
model[0].LoadModel("fridge/obj_files/fridge_body_hollow.obj");
model[1].LoadModel("fridge/obj_files/fridge_door.obj");
model[2].LoadModel("fridge/obj_files/fridge_drawer.obj");
model[3].LoadModel("fridge/obj_files/fridge_shelf_border.obj");
model[4].LoadModel("fridge/obj_files/fridge_shelf_border.obj");
model[5].LoadModel("fridge/obj_files/fridge_shelf_border.obj");
model[6].LoadModel("fridge/obj_files/fridge_shelf_border.obj");
model[7].LoadModel("fridge/obj_files/fridge_shelf_door_large.obj");
model[8].LoadModel("fridge/obj_files/fridge_shelf_door_small.obj");
model[9].LoadModel("fridge/obj_files/fridge_shelf_door_small.obj");
model[10].LoadModel("fridge/obj_files/fridge_shelf_door_small.obj");
model[11].LoadModel("fridge/obj_files/fridge_shelf_large.obj");
model[12].LoadModel("fridge/obj_files/fridge_shelf_large.obj");
model[13].LoadModel("fridge/obj_files/fridge_shelf_large.obj");
model[14].LoadModel("fridge/obj_files/fridge_shelf_small.obj");
lightModel.LoadModel("basic_cube.obj");

printf("*** %d models loaded ***\n", (int)(sizeof(model) / sizeof(*model)));

// Start all models from the centre
for (int i = 0; i < noModels; i++) {
	modelPosition[i] = glm::vec3(0.0f, 0.0f, 0.0f);
	modelRotation[i] = glm::vec3(0.0f, 0.0f, 0.0f);
}
```

Manually aligning the duplicated top shelf from the OBJ import:

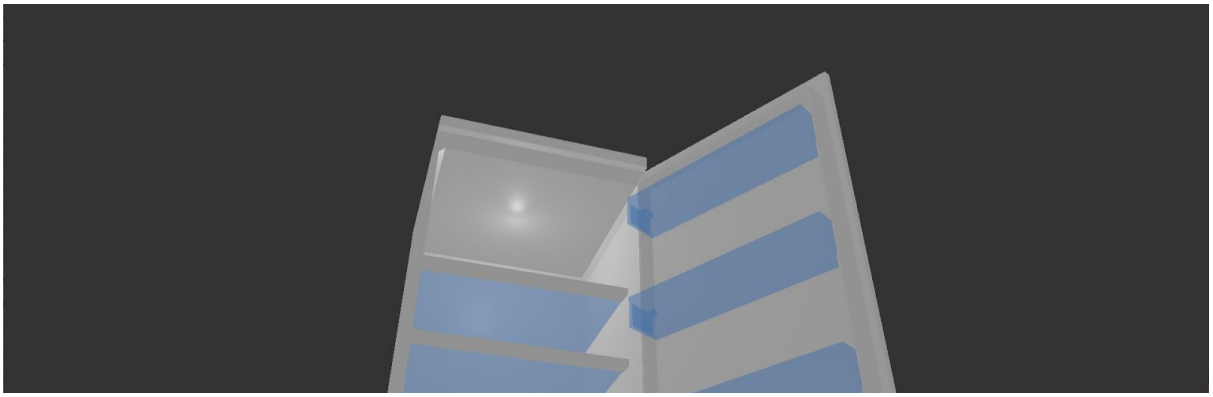
```
// Manually align middle shelves
modelPosition[4]  = glm::vec3(0.0f, -0.25f, 0.0f);
modelPosition[5]  = glm::vec3(0.0f, -0.50f, 0.0f);
modelPosition[6]  = glm::vec3(0.0f, -0.78f, 0.05f);

modelPosition[9]  = glm::vec3(0.0f, -0.35f, 0.0f);
modelPosition[10] = glm::vec3(0.0f, -0.7f, 0.0f);

modelPosition[12] = glm::vec3(0.0f, -0.25f, 0.0f);
```

Lighting and shaders

I used a simple **point light** for the interactive model placed inside the top of the fridge. It has a cube mesh associated with it for debugging, but the drawing of this has been disabled (commented out) for the final version. A simple white light colour is used:

```
GLfloat      ka = 1.5;                                // Ambient constant
glm::vec4    ia = glm::vec4(0.5f, 0.5f, 0.5f, 1.0f); // Ambient colour
glm::vec4    id = glm::vec4(0.5f, 0.5f, 0.5f, 1.0f); // Diffuse colour
glm::vec4    is = glm::vec4(0.5f, 0.5f, 0.5f, 1.0f); // Specular colour

Mesh         lightModel;
glm::vec3    lightDisp = glm::vec3(-0.369f, 0.652f, -0.332f);
```

These settings create the following light source:



The light source can be moved along the **x** and **z axis** using the **UP**, **DOWN**, **LEFT** and **RIGHT** arrow keys, and the **y axis** using **PAGE_UP** and **PAGE_DOWN**.

The vertex and fragment shaders used are the same as those given in lecture material.

Textures

I converted the single texture variable into an array of textures:

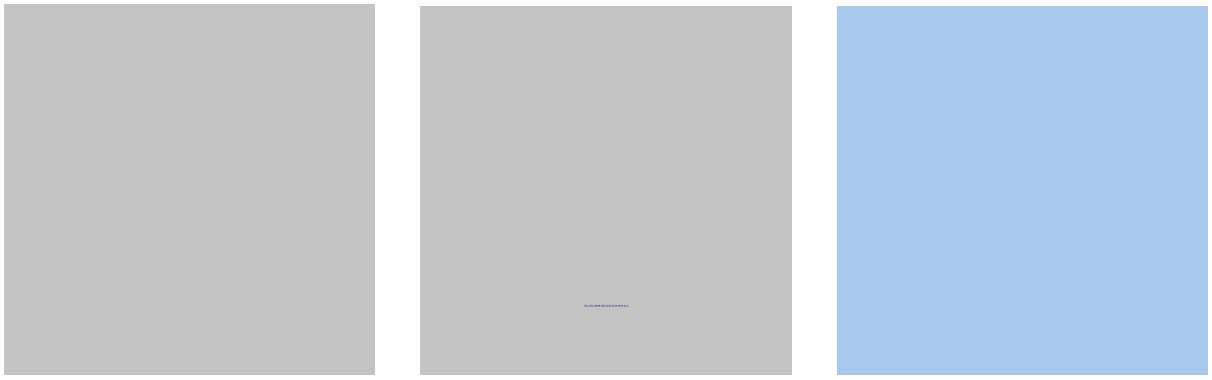
```
const int noTextures = 3;
string textureName[noTextures];
GLuint texture[noTextures];
int tex_location;
```

This allowed me to import multiple textures. Additionally, OpenGL's **BlendFunc** allows me to use a texture's alpha channel for transparency. I imported all of the textures into the texture unit so that I could bind the relevant ones when necessary:

```
// Load Texture OPENGL 4.3
textureName[0] = "fridge/uv_maps/fridge_body.png";
textureName[1] = "fridge/uv_maps/fridge_door.png";
textureName[2] = "fridge/uv_maps/fridge_trans_plastic.png";

// Generate the number of textures we need
glGenTextures(noTextures, texture);

for (int t = 0; t < noTextures; t++) {
	// Load image Information.
	stbi_set_flip_vertically_on_load(true);
	int iWidth, iHeight, iChannels;
	unsigned char* iData = stbi_load(textureName[t].c_str(),
		&iWidth, &iHeight, &iChannels, 4);

	// Load and create a texture
	// All upcoming operations now have effect on this texture object
	glBindTexture(GL_TEXTURE_2D, texture[t]);
	glTexStorage2D(GL_TEXTURE_2D, t+1, GL_RGBA8, iWidth, iHeight);
	glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, iWidth, iHeight, GL_RGBA,
		GL_UNSIGNED_BYTE, iData);

	...
```

I had to use **`stbi_set_flip_vertically_on_load(true)`** as STBI decided loading in textures upside-down was intuitive. It is however less efficient than if I manually flipped the texture files themselves.

My three textures are very simple, shown and described below:



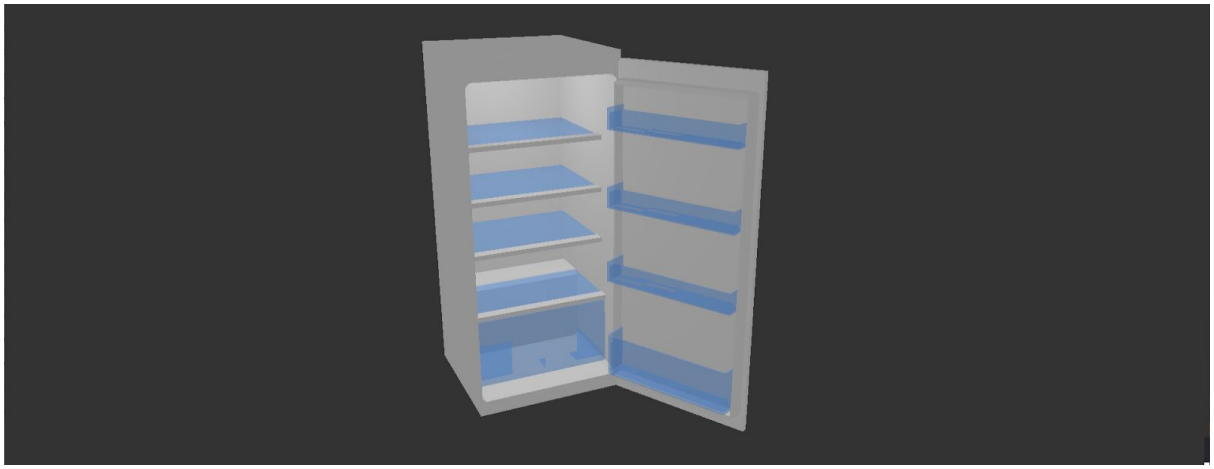
Left: *Main fridge texture / colour*

Middle: *The door texture, including fake logo*

Right: *Blue shelf texture with 0.5 opacity*

Within the render loop I cycle over all models in the array, and switch texture based on which model is about to be drawn. Re-binding textures is also not the most efficient way to accomplish this, but it is one of the simpler ways, and does not involve re-writing shaders:

```
// ...bind textures
 if (i == 1) {
        // ...if the model is the fridge door:
        glBindTexture(GL_TEXTURE_2D, texture[1]);
 }
 else if (i == 2 || i >= 7)
 {
        // ...if the model is a transparent shelf:
        glBindTexture(GL_TEXTURE_2D, texture[2]);
 }
 else
 {
        // ...otherwise, use a standard texture:
        glBindTexture(GL_TEXTURE_2D, texture[0]);
 }
```



Controls

The camera used is the standard FPS free-roam one provided by lecture material and is controlled using **W S A D**.

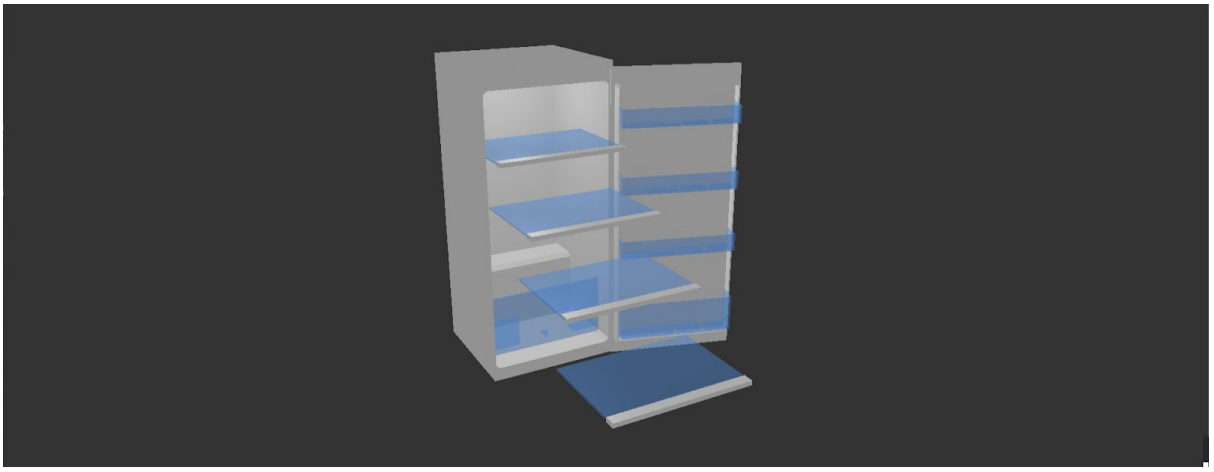
The light source can be moved along the **x** and **z axis** using the **UP**, **DOWN**, **LEFT** and **RIGHT** arrow keys, and the **y axis** using **PAGE_UP** and **PAGE_DOWN**.

Keys **1** and **2** open and close (rotate) the fridge door.

Keys **3** and **4** pull out and push in (translate on the z axis) the fridge shelves.

I have set **minimum bounds** for the door and shelves translation/rotation, and a **maximum bound** for the door rotation. Additionally, I have arranged the **shelves array** and used a **modulo** of the index to stagger the speed at which the shelves translate in and out of the fridge, producing a slightly more visually appealing interaction:

```
int doorPieces[5] = { 1, 7, 8, 9, 10 };
int shelves[8]    = { 3, 12, 5, 14, 11, 4, 13, 6 };

// Rotate door
if (keyStatus[GLFW_KEY_1]) {
    if (modelRotation[doorPieces[1]].y <= 1.0f) {
        for (int i = 0; i < 5; i++) {
            modelRotation[doorPieces[i]].y += 1.0f * cameraSpeed;
        }
    }
}
if (keyStatus[GLFW_KEY_2]) {
    if (modelRotation[doorPieces[1]].y >= -2.065f) {
        for (int i = 0; i < 5; i++) {
            modelRotation[doorPieces[i]].y -= 1.0f * cameraSpeed;
        }
    }
}

// Pull / Push shelves
if (keyStatus[GLFW_KEY_3]) {
    for (int i = 0; i < 8; i++) {
        modelPosition[shelves[i]].z += (1.0f + (i % 4)) * (0.2f * deltaTime);
    }
}
if (keyStatus[GLFW_KEY_4]) {
    for (int i = 0; i < 8; i++) {
        // We cap the backwards distance by shelf no. 6 because
        // it sticks out slightly
        if (modelPosition[shelves[7]].z >= 0.05f) {
            modelPosition[shelves[i]].z -= (1.0f + (i % 4)) * (0.2f * deltaTime);
        }
    }
}
```



The staggered shelves having been pulled out