COSC 450 ASSIGNMENT 2

1. Introduction

This report covers the implementation details, results and conclusions from adding X-ray rendering to the provided Augmented Reality program. The program was tested on the checkerboard target and videos from Assignment 1. The submitted code has some platform-specific changes reverted to the original format or the files with only those changes not included (e.g. Assignment2Main.cpp), such as include statements and filename formatting.

2. Edge Extraction

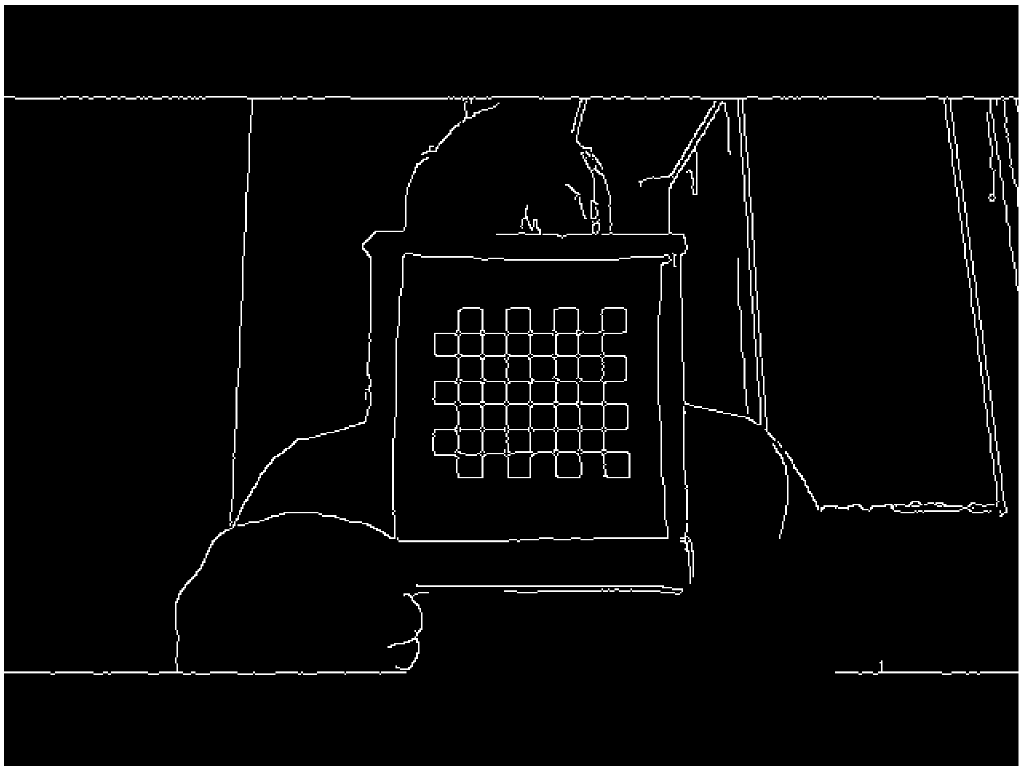

Figure 1 - extracted edges from checkerboard.mp4, displayed with imgshow

The above edges of a video frame were produced with Canny edge extractor. An initial threshold of 100 failed to extract the central checkerboard edges on subsequent frames. This was adjusted until the threshold argument value of 50 was found to capture all edges in the video.

The extractEdges method added converts a video frame to greyscale, extracts the edges and creates a binary edge mask from them.

Canny reference:
https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

3. X-Ray Rendering

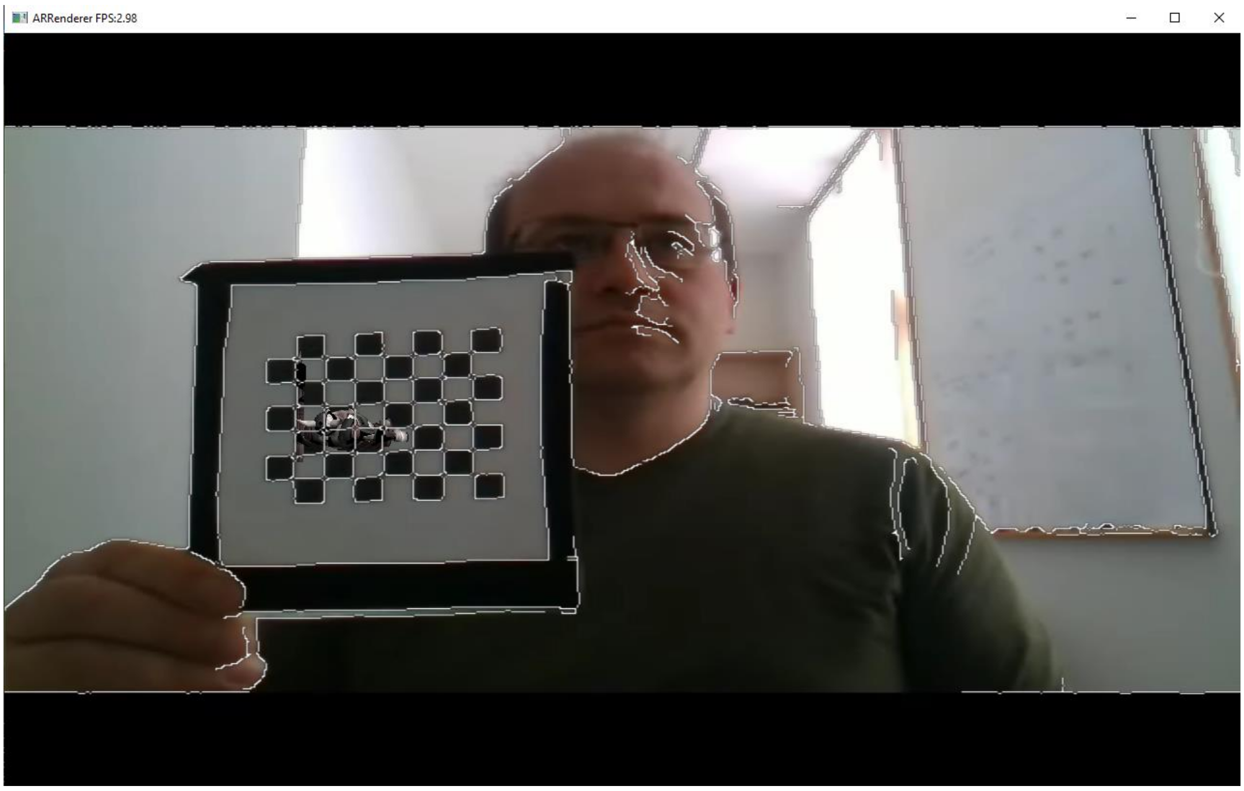

Figure 2 - binary edges rendered on top of model

The extractEdges method also creates an RGBA texture from the edges. The greyscale frame outputted from Canny is converted to an RGBA image with empty alpha channel. The channels are then split so the binary edge mask can be saved as the alpha channel.

Adding alpha channel reference: https://stackoverflow.com/a/50159769

The merged RGBA image is then used for the new texture, which is bound to the current OpenGL context (and has its texture ID returned from the method to aid in debugging). Though originally outside of it, the texture generation was moved to extractEdges to mimic the equivalent prepareTexture method, and retain the encapsulation structure of the program.

3-pass rendering is implemented, with blending enabled for the third pass and disabled for the first two. Color blending was initially used for the binary edges, but alpha blending and the adding of the alpha channels to both edge textures was implemented to simplify switching between white and coloured edge rendering.

Blending reference: https://learnopengl.com/Advanced-OpenGL/Blending

4. Variations on X-ray rendering

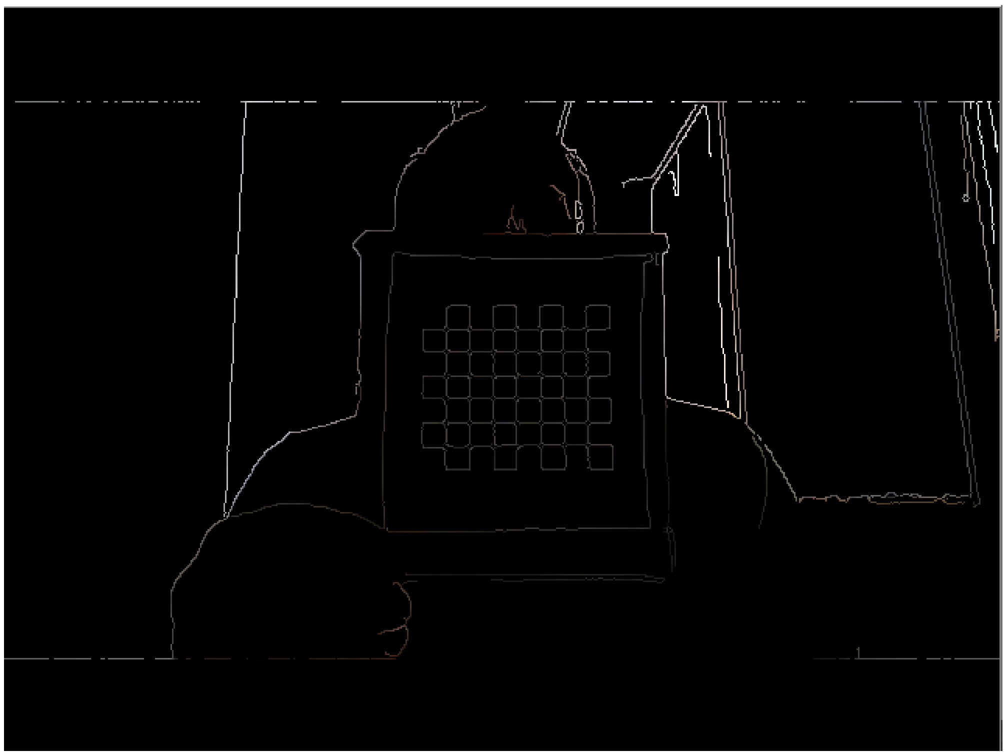
Figure 3 - coloured edge matrix displayed with imshow

The above edges were coloured by copying the original coloured frame to a new matrix with the binary edge mask applied. The transparent background was again represented in the alpha channel with the binary edge data.

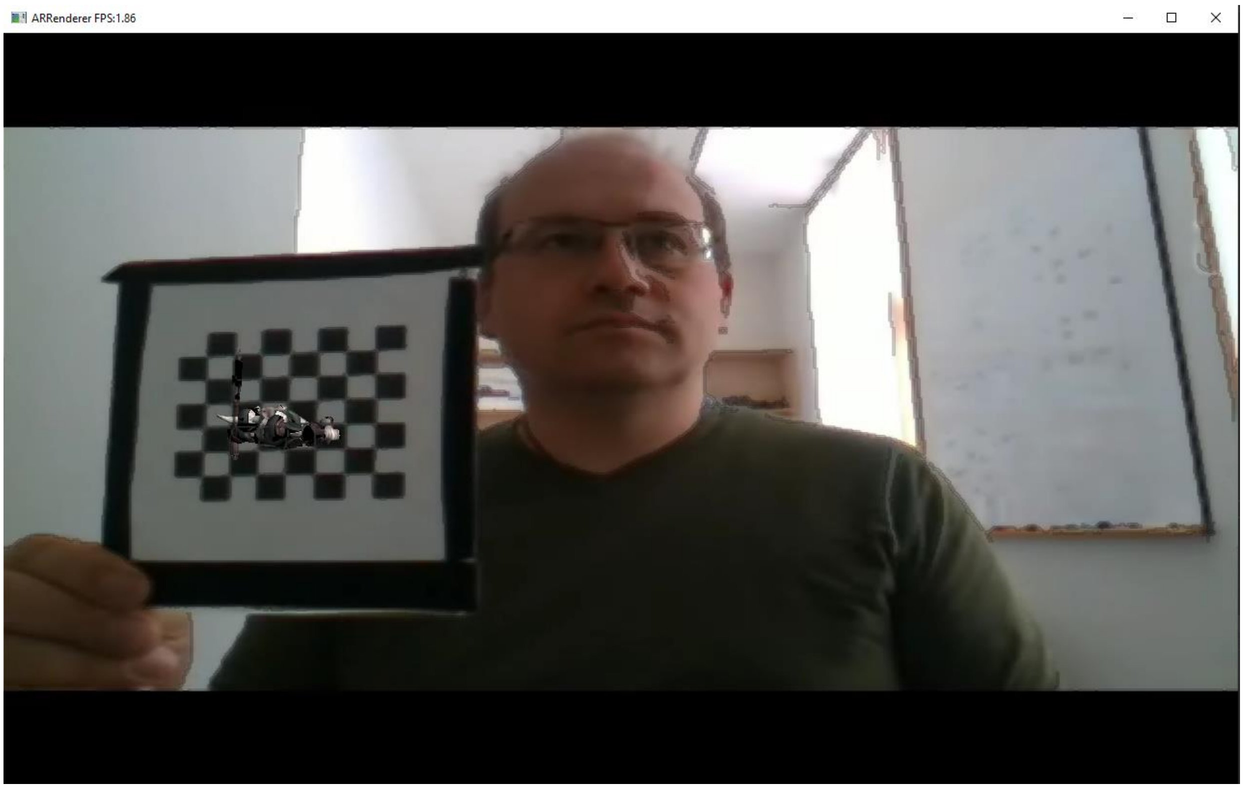

Figure 4 - coloured edges rendered on top of model

The coloured edges integrated into the background better than white ones, but had little contrast against the grey model chosen as default. This gave a fairly minimal impression of X-ray depth, although the effect was more convincing on the full render window than this compressed screenshot. A model with more colour contrast to the checkerboard may have shown the X-ray effect better, but the code to change the model loaded was unclear to me.

The FPS before adding the alpha channel and alpha blending was slightly lower on average, but both the earlier opaque and later transparent white edge renderings stayed within the 2-3FPS range. There was not noticeable difference in performance when colouring the edges.

NOTE: The submitted code uses coloured edges. If white edges are needed, line 1156 can be changed from:

```
    extractColorEdges(imageMatGL);
```

to:

```
    extractEdges(imageMatGL);
```

Dilation of the edges was attempted to see the effect of thicker or thinner rendered edges, with the intermediate code commented in the extractEdges method of the submission, but a working implementation could not be found.

Dilation reference: https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html

5. Discussion

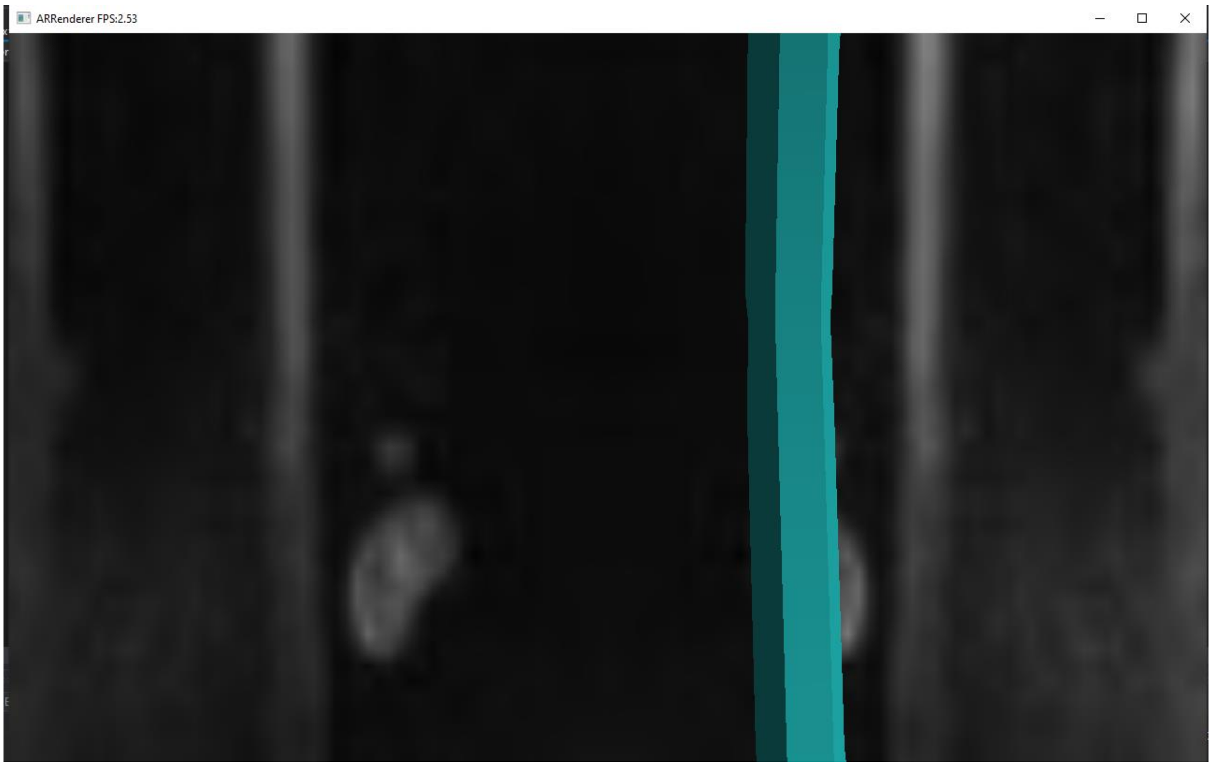

Figure 5 - rendering error occurring on the second virtual object pass, possibly a corrupted frame due to bad memory access

Due to an erroneous ampersand (required on my machine due to compiler differences and placed on a variable usage rather than its creation), the above error occurred in the base code though most of development.

```
        auto currentModel = it->second;
        if (currentModel.seennow) {
```

The optimal solution suggested was to replace the pointer designation of currentModel with auto typing. Interestingly, in the extensive search for the error (including hardware checking, revising the edit history of all files, attempts at different Visual Studio project and Cmake setups, disabling various rendering passes to isolate the error, and stepping through breakpoints), another bug was found.

```
if (!import3DFromFile(modelName, models[markerNum].scene, models[markerNum].importer,
models[markerNum].scaleFactor))
            return(-1);
```

This original code returns the equivalent of a true boolean in C++, when the intent was to return false to trigger the "Could not load the model" error message. The successful return value was 0, triggering the error message incorrectly. This was corrected to failure returning 0 (false) and success returning 1 (true).

Comparing the two methods of edge rendering implemented, a trade-off was clearer than any superior method. Plain white edges rendered straight from the binary edge mask provided excellent contrast for the X-ray effect, causing the model to appear to be rendered within the checkerboard. This would be especially apparent in compressed images such as those in this report or on small/distant screens. However, the white edges gave a highly stylized look to the background image, reducing photorealism in the scene especially on large/close screens. This method could be optimal in contexts where colour contrast is low but a strong X-ray effect is needed, and the video background is not in strong focus of the user.

Colouring the edges reduced the X-ray effect on a model with similar colouring, but gave a much more realistic look to the video background. For models contrasting with the edge colours, I predict edge colouring would be the superior rendering technique.

A potential operation to improve the X-ray effect while maintaining background appearance would be to apply a gradient mask onto the edges generated from the rendered virtual object's outline. This would cause edge transparency to decrease smoothly outside of the object to integrate the edges in that region into the rest of the (edgeless) scene.