CITS3003 Graphics and Animation
Project Part 2 Report

Bruce How (22242664)

May 23, 2019

1 Overview

The submitted code covers all the required project functionalities. Contribution breakdown is not required as I have worked individually.

I had several rare occurrences with the "Fatal Error: Out of Memory" error message that would occasionally crash my scene. I was unable to reproduce the error as it appeared very random. To combat this issue, I added a few conditional checks to ensure that memory is not used when unavailable.

```
if (nObjects >= maxObjects) return; // Potential fix to the Fatal Error: Out of
    Memory
```

2 Implementation

A Texture Scaling

Texture scaling was somewhat already half-implemented in the first part of the project. To fully implement texture scaling, the *texScale* variable had to be added to the fragment shader. This variable will be multiplied on top of the default 2.0 value.

```
gl_FragColor = (color * texture2D(texture, texCoord * 2.0 * texScale)) +
    vec4(specular / lightDropoff + specular2, 1.0);
// PART 2A. Multiply by texScale value
```

B 3D Human Modeling with MakeHuman

All three human models were generated using the *MakeHuman 1.0.2* software and exported to the **mhx** (Blender Exchange) format. These include *ElderlyMan.mhx*, *MatureWomen.mhx*

and *YoungBaby.mhx*. Variations in their parameters were used resulting in three very different human models.

I was unfortunately unable to import these *.mhx* files into Blender. After some researching, I found that the old *.mhx* format is deprecated and replaced with *.mhx2* as of *MakeHuman 1.1.x*. This is important as the MakeHuman plugin for Blender no longer supports *.mhx* files, but supports *.mhx2* files.

C Animating the 3D Human in Blender

I followed the provided steps and downloaded three *.bvh* animation files, an old-man walk animation, cartwheel animation, and a baby crawl animation; one for each human model. These *.bvh* files were then combined together with its respective *.mhx2* human model and exported to DirectX. I initially had trouble exporting the files to DirectX as the export option was not provided by Blender. I was able to overcome this issue by installing the "*Import-Export: DirectX X Format*" add-on.

After exporting to *DirectX*, I was initially unable to get my model animations working and facing the right way up. The cause of the issue was because the default *DirectX* export options did not include the animation aspects of my model. This issue was fixed after re-exporting my models with the appropriate *DirectX* export options.

D Animation

D.1 Coding Animation

I initially came across an issue with the default scale of my models. The sample model *model-56.x*, which was used to test my animation code, was extremely small compared to the rest of the objects in the scene. The following code was added to the *addObject()* function in the cpp file in order to scale up the additional models and set the model frames.

```
if (id == 56) {
    sceneObjs[nObjects].scale *= 10; // PART 2C. Scale the human models by 10
    sceneObjs[nObjects].frames = 300;
    sceneObjs[nObjects].hasAnim = true;
}
```

Additional variables were added to the *SceneObject* struct that were used to appropriately display the animations.

```
// PART 2D. Animation variables
bool hasAnim; // If the obj has animation
int frames; // The number of frames
```

The below segment of code was added to the *display()* function and handles the calculation for the *poseTime* and the animation itself. This variable is used in the *calculateAnimPose()*

function that handles the animations.

```
float poseTime = 0.0;
if (sceneObj.hasAnim)
    poseTime = (int) POSE_TIME % sceneObj.frames;
```

A few constraints were put in place to validate the animation speed. *ANIMSPEED* is a global variable that controls the animation speed. It is bounded to have a lower limit of 0.1 to ensure that the animation is not completely frozen, and an upper limit of 10, so that the max animation speed is at a reasonable.

```
// Set bounds for the animation Speed
if (ANIM_SPEED > 10) ANIM_SPEED = 10;
if (ANIM_SPEED < 0.1) ANIM_SPEED = 0.1;

POSE_TIME += ANIM_SPEED/100;
```

D.2 Animation Speed

Users are able to control the animation speed by using their mouse. Similar to the previous parts, this was implemented with the use of the *toolCallback* functions, and an additional function *adjustAnimSpeed* as shown below.

```
// PART 2D. Adjust animation speed
static void adjustAnimSpeed(vec2 an_sp) {
    ANIM_SPEED += an_sp[0];
}

if (id == 62) { // Animation speed
    setToolCallbacks(adjustAnimSpeed, mat2(1, 0, 0, 10),
                    adjustAnimSpeed, mat2(1, 0, 0, 10));
}
```

D.3 Pause/Resume All Animations

Pausing all animations was achieved simply by setting the *ANIM_PAUSED* variable to true. Similarly, all animations will be resumed once the this variable is set to false. The idea behind this is that *POST_TIME* is conditioned to only increment in value if all animations are not paused. The following piece of code from **D.1** was modified as shown below.

```
if (!ANIM_PAUSED) {
    POSE_TIME += ANIM_SPEED/100;
}
```

D.4 Smarter Animation Menu

A submenu was used to house the set of animation controls to reduce the clustering of menu items. These controls include *Resume Animation*, *Pause Animation*, and *Animation Speed*. Much like the extension part for Part 1, these controls are conditioned to display only when applicable. For example, the *Resume Animation* will not be displayed if the all the animations are already playing.

```
static void animationMenu(int id) {
    if (id == 60) { // Animation resume
        ANIM_PAUSED = false;
        makeMenu();
    }
    if (id == 61) { // Animation pause
        ANIM_PAUSED = true;
        makeMenu();
    }
    if (id == 62) { // Animation speed
        setToolCallbacks(adjustAnimSpeed, mat2(1, 0, 0, 10),
                        adjustAnimSpeed, mat2(1, 0, 0, 10));
    }
}

// PART 2D. Animation control using a sub-menu
int animationMenuId = glutCreateMenu(animationMenu);
if (ANIM_PAUSED) {
    glutAddMenuEntry("Resume Animation", 60);
} else {
    glutAddMenuEntry("Pause Animation", 61);
}
glutAddMenuEntry("Animation Speed", 62);
```

3 Experience

In general, I found the second part of the project to be much more manageable in terms of the amount of coding required. This part of the project has introduced me to the basics of animations and modelling. Modelling in Blender proved to be a difficult task due to my lack of experience with Blender, but was not difficult to learn. Overall, this part of the program is a necessary and valuable lesson which has provided me with the needed experience in animation that would definitely be useful in the future.