CITS3003 Graphics & Animations

PROJECT 2019

Alexander Varano della Vergiliana | Student ID#: 22268701 | May 1, 2019

Building the Program

This implementation of Project 2019 incorporates all the functionality specified in the project document for parts 1 and 2, as well as two additional features which are discussed in the 'Additional Functionality' section of this report.

Being initially presented with the skeleton code, several header files and the assimp library, the main steps taken to solve the problems throughout the project were very much akin to a process of reverse engineering, with reflection on tasks undertaken in the tutorial labs. Starting by working through and understanding the scene-start.cpp file, then working back to the header files gnatidread.h and gnatiread2.h, understanding how the functions were called and variables set provided a very good basis for implementing the remaining features required in the project. When it finally came to the animation implementation, working through and understanding the functionality assimp library was critical to implementing the final features of the project.

This process of reverse engineering and an understanding of the various topics reviewed over the semester allowed for a strong understanding of the default implementation and how it could be added to and modified to incorporate the features requested throughout the project.

Challenges Encountered and Resolutions

Animation: One of the main bottlenecks encountered during the development of the solution were in the final stages when it came time to incorporate the animated models within the code. Reflecting back on past tutorials provided some guidance in how to proceed, utilizing combinations of the glutGet(GLUT_ELAPSED_TIME) function in conjunction with a floating point variable could produce a viable value to be passed as the pose_time argument. The decision to add an additional variable to the SceneObject structure was made which would record the time an object was added to the scene (int timeAdded). The addObject() function includes a line at 307 where we record this time. Subtracting this timeAdded variable from the current GLUT_ELAPSED_TIME, multiplied by a value of 0.001 produced good initial results in being able to animate the model in the scene, however only once with no looping.

Looping this animation presented the bigger challenge. It was initially obvious that to loop the animation we needed to know its duration in terms of frame numbers. However how we get and pass that value back into scene-start.cpp required a detailed analysis of how the assimp library worked. After some research it was evident that the double variable mDuration from the aiAnimation structure defined in anim.h of the assimp library is what was required.

This final implementation for each animated model is illustrated in the following figure and the full solution is found in the drawMesh() function at line 458.

```
457     //calculate pose time, new addTime element added to Object struct.
458    //CHANGED: PART 2 D.
459     float pose_time;
460     float animDuration;
461
462     if(sceneObj.animSpeed <= 0)
463     {
464       sceneObj.animSpeed = 0.001;
465     }
466
467     //Dancing Lady - animation loops forward & backward
468     if(sceneObj.meshId == 57)
469     {
470        animDuration = static_cast<float>(scenes[57]->mAnimations[0]->mDuration);
471        pose_time = fabs(animDuration - ((glutGet(GLUT_ELAPSED_TIME) - sceneObj.timeAdded) *
472                    fabs(sceneObj.animSpeed)));
473        if(pose_time >= animDuration)
474        {
475          sceneObjs[3].timeAdded = glutGet(GLUT_ELAPSED_TIME);
476        }
477     }
478     //Dancing Baby - animation loops forward & backward
```

Figure 2: Animation code illustrating looping forward and backward functionality

Note in the above code that the MeshIds (ie. 57) are explicitly mentioned when assigning the animation duration times to the animDuration variable:

animDuration = static_cast<float>(**scenes[57]**->mAnimation[0]->mDurction);

Effort was made to be able to reference the current mesh dynamically using sceneObj.meshId, as this would provide much more flexibility in the scene editor when adding animation, however I wasn't successful and trying to implement this would result in segmentation faults at compile time. As a result, the three animated models that appear at run time are accounted for in the code from line 459, providing the ability to loop forward and backwards continuously. However, any additional animation added by the end user will result in only one pass of the animation forward with no looping capability.

Additional Functionality

Two additional features outside of the required specification have been added to the project implementation:

- Keyboard movement of the view
- Ability to delete the current object

Keyboard movement: Functionality to move the view via the ‘a’, ‘d’, ‘s’, ‘w’ keys has been implemented to allow for greater freedom of movement around the 3D world in combination with the mouse. The keys ‘s’ and ‘w’ move the view closer and further away, much like the scroll wheel of the mouse. The keys ‘a’ and ‘d’ provide new movement options in the ability to strafe left and right. The code for this new functionality can be found in the keyboard() function at line 823. A new global variable ‘camStraif’ is utilized, which is applied to the view transformation calculation in the display() function at line 537.

Ability to delete current object: Functionality to delete from the scene the current object defined by the variable ‘currObject’ has been implemented as a menu function (menu ID 40). A new function has been defined at line 317 called deleteObject(). This function clears the meshId and TexId structure variables with NULL and decreases the toolObj, currObj and nObjects variables by 1.

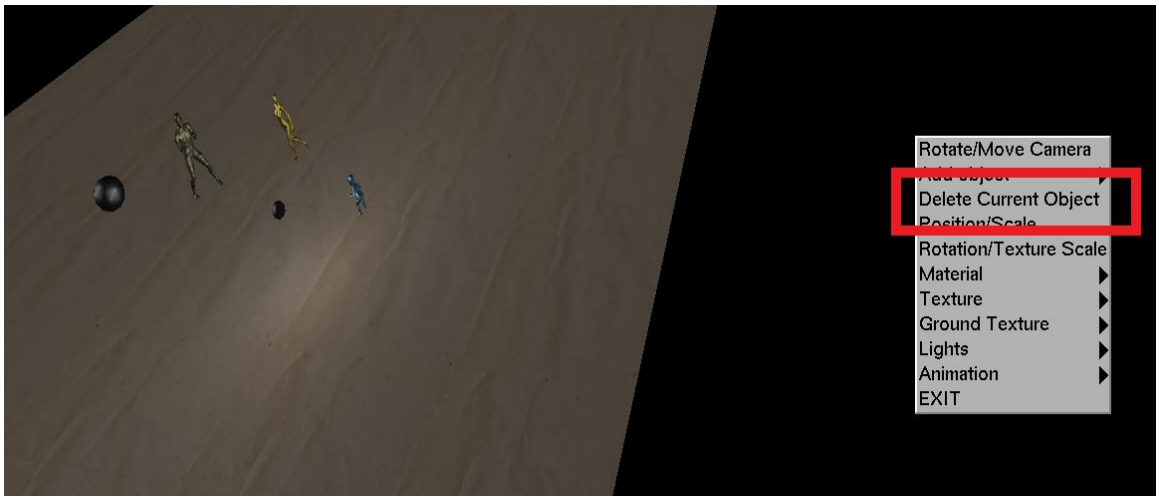

Figure 1: Strafing (to the right illustrated) and new ‘Delete Current Object’ features

Project Reflection

Overall, I found this project to be quite enjoyable and informative. The project incorporated the broad range of topics we encountered during the semester whilst without being completely overwhelming from a time management perspective. The tasks were challenging and diverse, plus the visual feedback you got from completing the tasks after each compile and at run-time was great and provided a good level of encouragement throughout the project.