Comp 465 - Warbird Documentation

Authors: Chris Bowles, Josh Nitzahn
12/10/13

There is include465.h, which gives the professor's tools for the class.
glmUtils465.hpp is for displaying glm vectors and matrices.
texture.hpp is for all functions involving vectors.

The program uses a single class for 3D objects, called Object3D.

Object3D has many functions for setting and retrieving values. The model matrix does not change until update() is called. The update() function also recursively calls updates for any Object3D that is set to be a satellite. Satellites can be added during runtime.

Each Object3D contains the following:

- number of satellites
- whether it is a camera or not (boolean)
- orbital rotation matrix
- regular rotation matrix
- orbital translation matrix(orbit distance)
- regular translation matrix
- an axis of orbit, in world space
- an orbital rotation increment
- eye, at, and up vectors for the view matrix
- a view matrix defined by the current orientation

Object3D has two constructors, one for allowing initialization of satellites
The destructor destroys any satellites along with the object.

There are functions for retrieving the number of satellites, the model matrix(after update), the regular rotation matrix, the view matrix, and the current position(based on last model matrix update).

There are functions for setting the position to orbit around and the increment to orbit (in radians).
The orient function is used only by missiles. It takes two arguments for the right and at vectors to orient the missile. It calculates the cross product for up.
The reset() function takes a matrix and sets the regular rotation matrix to its orientation. If the Object3D is not a camera, it will also set the regular translation to the matrix's position.
move() translates the object on local coordinates
move2() translates on global coordinates
The functions yaw(), pitch(), and roll() rotate the regular rotation matrix along the appropriate axis.

DEFAULT_CAMERA values are for the initial view in the window. Without them, the view matrix for the camera is an identity matrix and will not work for OpenGL.

Other #define declarations are for easier editing.

nModels = how many model files are loaded.

The enum type for movement has up for tilting upward and down for downward tilting.
A missile in the notFired state is inactive and not drawn.
A tracking missile will track to nothing when the target is outside of detection radius.

The planet satellites are given a translation vector that tells where to start orbiting.
The scale matrices are set using the bounding radii from the loadTriModel() function in include465.h

The ship has a camera satellite.

The title bar updates every second, except for the current view, which changes immediately.

Gravity formula is directionVec * -1 * (GRAVITY/distance^2)

The ship moves on local axes for regular movement and gravity pulls on global axes.

The objects are all updated before collision detection and missile tracking.

proximityCheck() returns true if the first two arguments are within the distance in the third argument.

The track() function puts priority on the second target. It will orient a missile to the second target if both are in range.

In total, there are 2 source files: 3dobject.hpp and warbird.cpp. We also have the two glsl files for shaders, unfortunately, we did not have enough time to implement the usage of our texture (stars.raw) into the program, despite our successful attempt at loading the texture into the program. The .tri files used are missile.tri, launcher.tri, ship2.tri, and planet.tri.

As a group, we used a Github repository to manage our progress individually. This made it extremely easy to commit changes to the code, as well as to visualize what specific edits were made in the code for a given task.

In addition, we used the .dll files for compilation in the same directory as our source files for simplicity purposes. When developing this code, we used a .bat file to automatically compile the code. Unlike normal projects, this was done entirely though the usage of MinGW (g++) through the command line terminal. We did not use Visual Studio at all!!

In theory, this code should be able to import into a fresh Visual Studio project and compile right then and there. However, if it does not, you may compile the code in the same fashion that we did by downloading the portable version of MinGW that we were using. (See the .txt file with the mediafire download link…) Make sure to put the extracted MinGW folder in the same location that the includes465 and warbird directories are. From there, double click warbird's "warbird.bat" file to compile and run the program. (The command line will pause right before execution in case errors occurred, but all it will display on the command line are deprecated string warnings. Ignore these.)

As mentioned, the only feature we were not able to implement was the shader/texture part. Other than that, all of the “game logic” works, as well as the complete missile functionality. If you run into problems getting the program to execute, please feel free in contacting either of us (Josh or Chris) immediately.