CPSC 436D: Video Game Programming
Intro to Game Graphics Assignment

Due: 23:59 PM, Friday January 18, 2019

1 Introduction

The goal of this assignment is to introduce you to basic graphics interface programming. You will experiment with rendering, shaders, and event-driven frameworks in general.

In the assignment you will implement a simple 2D game where the user controls a salmon swimming upstream. Can your salmon dodge the turtles that rush by? How many fish will you eat ? You will implement this game by building on top of an instructor-provided template, adding the required code

The assignment includes both a required (80%) and a free-form component (20%). The goal of the latter is to let you experiment with computer graphics and have fun.

2 Template

The template code provides a starting base for your work. You will find comments throughout the files to help you guide in the right direction. The directory is structured as follows:

- The directory `src` contains all the header (`.hpp`) and source (`.cpp`) files used by the project. The entry point is located in `a1.cpp` while most of the logic will be implemented in `world.cpp` together with the respective salmon, fish and turtle `.cpp` files.
- The `data` directory contains all audio files, meshes, and textures used in the code.
- The directory includes a Makefile and a Visual Studio project in the `visual_studio` directory.
- The external dependencies are located in the `ext` subdirectory, which is referenced by the project files, it contains header files and precompiled libraries for:
  - `gl3w`: OpenGL function pointer loading (Header-only)
  - `GLFW`: Cross-platform window and input

- `SDL/SDL_mixer`: Playing music and sounds
- `stb_image`: Image loading (Header-only)

2.1 Transformations and Rendering

The template uses modern OpenGL and in order to keep similar transformation semantics as in `glPushMatrix() glTranslate() glRotate() glScale() glPopMatrix()` the following functions are provided:

- `transform_begin()`: Resets the current transform to identity
- `transform_rotate()`: Applies a rotation matrix to the current transform
- `transform_scale()`: Applies a scale matrix to the current transform
- `transform_translate()`: Applies a translation matrix to the current transform
- `transform_end()`: Signal that you are finished with transformations. The obtained `3x3 transform` is the passed to the Vertex Shader and multiplied by the orthographic projection matrix.

Be careful about the order of transformations as they are being multiplied **before** being passed as uniform data to the shaders.

3 Required Work (80%)

1. Getting started (15%):

   (a) Download and untar the source template, template.zip. It should match the structure specified in the Template section. It also contains an mp4 video demonstrating how your solution should look like once the required parts are completed. The package can be downloaded from:

   ```
   git clone https://stash.ugrad.cs.ubc.ca:8443/scm/cs436d18w2/assginment1.git
   ```

   (b) Play the `solution_demo.mp4` to get a sense of what a possible assignment solution should look like. The speed of the water current can be controlled with `< (shift + ,)` and `> (shift + .)` keys (as it can be seen toward the end of the demo).

   (c) Build the template using CMake.

   If you are using Windows, you can skip this step of installing 3rd party libraries. For OSX users, here are the commands for Homebrew (`https://brew.sh/`):

   ```
   brew install pkg-config
   brew install glfw3
   brew install sdl2
   brew install sdl2_mixer
   ```

And for MacPorts (`https://www.macports.org/`):

```
port install pkgconfig
port install glfw
port install libsdl2
port install libsdl2_mixer
```

For Linux users, please install `glfw3`, `sdl2` and `sdl2_mixer` using your package manager.

If CMake is not already installed, you can download and install from the CMake website: `https://cmake.org/download/`; or use the package manager of your choice.

Create an empty directory as the build directory, which we assume is named `build`. You can configure the project using CMake GUI or the command line. For the GUI, enter the assignment template folder (which should contain a CMakeLists.txt file) as Source and the `build` folder as the Build. Then, configure, and if the configuration is successful, generate. For the command line, cd inside the `build` and run:

```
cmake [path_of_assignment_template] -DCMAKE_BUILD_TYPE=[Debug|Release]
```

Now you can build the generated project using make, Visual Studio, etc. For Visual Studio users, change the working directory ([Debug] → [436d Properties] → [Debugging] → [Working Directory]) to `$(ProjectDir)$(Configuration)` so you can launch the game using the VS debugger.

Note that for this specific project, you need to rerun CMake configuration and generation if you move the template folder.

2. For a basic version of the game make the following changes to the provided template (65%):

   (a) Movement: pressing the Up/Down directional keys should make the salmon swim up and down and pressing the Left/Right directional keys should make it swim left and right. The keyboard callback function is located in `World::on_key()`. Use it to keep track of the state of the keys or Salmon's velocity. You can then use it in `Salmon::update()` to correctly update its position calling `Salmon::move()`. In order to render the salmon in the correct location you will need to modify `Salmon::draw()` and issue the correct `transform_translate()` command.

   (b) Rotation: Provide mouse control for rotating the salmon, so that moving the mouse to the left/right rotates the salmon clockwise/counterclockwise. You can obtain the mouse position in the `World::on_mouse_move()` in window-coordinates, relative to the top-left of the screen. You can calculate the rotation angle with respect to its default facing direction (positive X axis), which can then be updated using `Salmon::set_rotation()`. As for the position, In order to render the correctly orientated salmon you will need to modify `Salmon::draw()` and issue the correct `transform_rotate()` command.

(c) There are two other type of entities in the sea: turtles and fish. By reading the initialization and rendering code for the different entities you will notice that the entities are rendered in two different ways: The salmon has a more complex geometry and each vertex has its own color, while the turtle and fish are 'faked' using a texture, which is applied on a quad (two triangles). The turtles are dangerous for the salmon, while the fish can be eaten by the salmon in order to obtain points. Points are then displayed in the window title. While the collision code is already implemented in `Salmon::collides_with()`, you need to properly handle the reactions with:

i. Turtle: If a collision with a turtle occurs the `Salmon::kill()` method is called. You need to update the salmon's alive state and also change its color. See `Salmon::draw()` to understand how the color variable is passed to the shader and `colored.fs.glsl` to see how it's being used to modify the final salmon color. Then modify it to make the salmon red after a collision. You should also see him sink down if you have properly updated its alive state.

ii. Fish: Whenever a Salmon eats a fish, the score should be updated and the salmon should temporarily light up. The method `Salmon::light_up()` starts the countdown for the duration of the light (`Salmon::m_light_countdown_ms`). The salmon is lit up in the `colored.fs.glsl` shader based on its state which is passed as uniform from `Salmon::draw()`. For this question you need to pass the correct state to the shader only if the salmon has eaten recently and change the light color to yellow inside the shader.

(d) The underwater effect is achieved using a second-pass shader. The two-pass rendering code is provided. Your job of this part is to modify the water fragment shader, `shaders/water.fs.glsl`, for the underwater distortion and color shift. Note that you don't need to match the solution video exactly.

Hints for the distortion part (`distort`): think about the translation, what if the offset value is not uniform at all pixel locations but is varying like a wave function? What if this wave function is varying based on time? Another helpful piece of information is that the input and output values of `distort` are in $[0, 1]$, you should set the offset values to the right scale.

Hint for the color shift (`color_shift`): check the function `fade_color` in the same file. You may want to shift the underwater world slightly to blue.

If you are interested in how the two-pass rendering is done, the explanation is the following. In the first pass, the screen is rendered to a off-screen texture (see `World::draw()`). In the second pass a fragment shader is used to apply additional effects to each pixel of the texture obtained in the first pass. This is achieved by rendering a full-screen geometry in a similar fashion to how the turtles and fish are rendered (see `water.cpp`).

4 Creative Part(20%)

The required code changes described so far will let you earn up to 80% of the grade. To earn the remaining 20% as well as possible bonus marks you need to make the game more appealing (the size of the bonus will be at the marker's discretion). You could add features such as:

(a) make the salmon "bounce" off the top and bottom walls when it runs into them

(b) change the movement of the salmon to be consistent with its orientation, so that the up/down keys move the salmon along the direction it is aligned with

(c) give the salmon momentum so that it continues moving even when no keys are pressed

(d) improve the collision mechanism: As the salmon has a more complex geometry, it can be used to improve the accuracy by checking the collision against every individual triangle.. considering rotations?

(e) add additional visuals, either on the salmon or turtle (these can be animated or static)

(f) randomize the turtles' paths

(g) diversify the types of obstacles floating down the river

(h) did someone say bubbles?

Use your imagination to make any other changes, however please make sure you focus on tasks involving OpenGL knowledge.

To support both basic and advanced visualization and control features, you need to add a toggle option where the user switches between the two modes by pushing the 'a' and 'b' keys ('a' for advanced mode and 'b' for basic mode).

Document all the features you add in the README file you submit with the assignment. Advice: implement and test all the required tasks first before starting the free-form part.

5 Hand-in Instructions

1. You do not have to hand in any printed code. Create a README.txt file that includes your name, student number, and login ID, and any information you would like to pass on to the marker. Create a folder called "a4" under your "cs436d" directory. Within this directory have included all your source, data and make files as present in the template. We need to setup

2. The assignment should be handed in with the exact command:

```
handin cs436d a1
```

This will handin your entire a1 directory tree by making a copy of your a1 directory, and deleting all subdirectories! (If you want to know more about this handin command, use: man handin.) You can also use the web interface on your myCS page to upload the assignment.