Final Project Documentation

Jacob Williamson

Professor Burlick

CS 432 – Interactive Computer Graphics

Summer 2021

Introduction

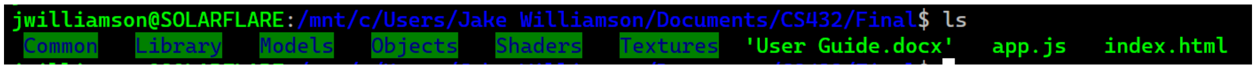
This project was created using WebGL 2.0. The world I created includes a small barn located on a flat, grassy rabbit farm. This farm specializes in Rainbow Rabbits because normal colored rabbits are boring. Additionally, the owners of the farm built a peculiar barn without an entrance. While on the surface it seems boring and stupid, they had a good reason. Inside, they're hiding a shiny, golden cow that's worth millions of dollars!

User Guide

Starting up

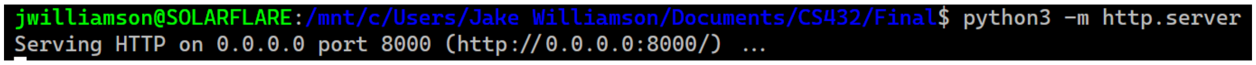
Dependencies: Python3, recent version of Chrome

1. Extract the contents of the Zip file
2. Navigate to the root of the project in your shell. Contents should look like:



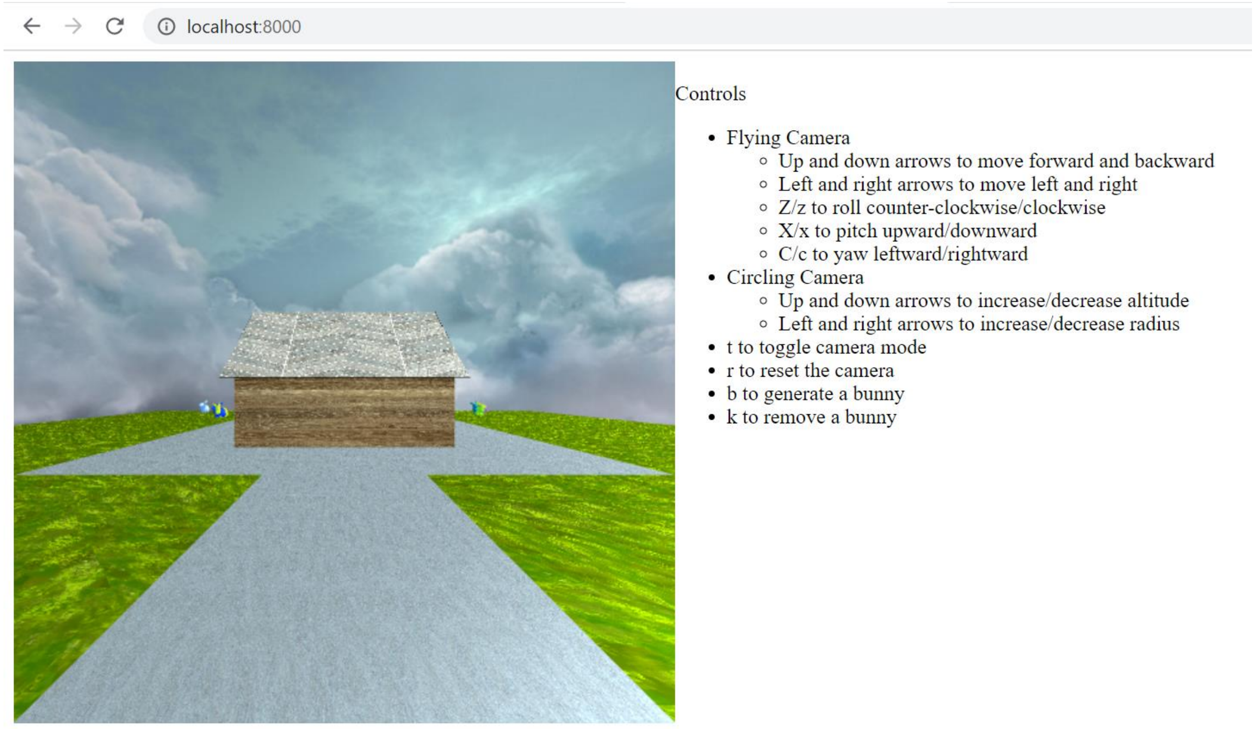
```
jwilliamson@SOLARFLARE:/mnt/c/Users/Jake Williamson/Documents/CS432/Final$ ls
 Common   Library   Models   Objects   Shaders   Textures  'User Guide.docx'   app.js   index.html
```

3. Run the command "python3 -m http.server"
   (Note: Python3 might have another name on your computer)
   Your shell should say:



```
jwilliamson@SOLARFLARE:/mnt/c/Users/Jake Williamson/Documents/CS432/Final$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

4. Navigate to "localhost:8000" in Chrome. You should be greeted with this:



Controls

- Flying Camera
  - Up and down arrows to move forward and backward
  - Left and right arrows to move left and right
  - Z/z to roll counter-clockwise/clockwise
  - X/x to pitch upward/downward
  - C/c to yaw leftward/rightward
- Circling Camera
  - Up and down arrows to increase/decrease altitude
  - Left and right arrows to increase/decrease radius
- t to toggle camera mode
- r to reset the camera
- b to generate a bunny
- k to remove a bunny

5. Explore!

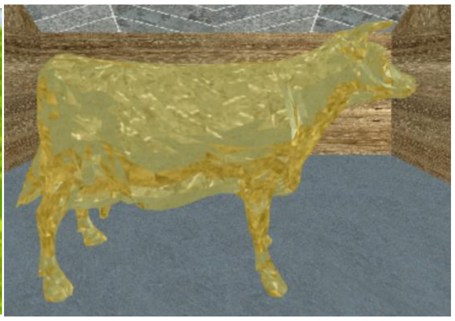
How to Navigate the World

- Using the arrow keys, Z/z, X/x, and C/c, you have all your need to move around the farm!

- If you'd rather have a bird's eye view, press 't' for a more automated fly-through! You can adjust your altitude with Up/Down and your radius with Left/Right
- If you want to go back to where you started, press 'r'!

Sights to See and Things to Do

- Rainbow Rabbits! This rabbit farm has tons of fun-colored rabbits to run around with!
    - Have you ever wanted to capture a rabbit? Try using your mouse to click on one at night! Rabbits are too alert during the day to be captured.
    - Want more? Press 'b'!
    - Too many? Press 'k'!
- Prized Golden Cow! Poke your head in the barn to find the secret shiny golden cow! It's so shiny that it reflects everything around it!



Technical Details

Rabbits

At the start, 10 rabbits are generated in the scene. This is done by creating a Bunny object and passing in random color details, so that each rabbit is unique. The vertex information is read in from a SMF file. Then, the rabbit's size and location are set using a model matrix.

Every so often, the rabbit will hop. This is managed by generating a random number each time the scene is rendered. If the generated number equals the "magic number" then the rabbit will hop. The direction, height, and distance of the hops are random. Hop height ranges from 1.5 to 7 and hop distance ranges from 2 to 7. Once the parameters of the jump are decided, then the distances in the X direction and the Z direction are calculated using the following formula:

- DX = distance * sin(direction)
- DZ = distance * cos(direction)

For each time the scene is rendered after the rabbit starts its jump, its location and height will be updated according to the following formula:

- newX = currentX + (1 / maxStep) * DX
- newZ = currentZ + (1 / maxStep) * DZ
- newY = DY * sin((currentStep / maxStep) * PI)

Where maxStep is the number of renders it will take to complete the jump (default 250) and currentStep is the number of renders since the start of the jump. DY is the height of the jump.

While not quite qualifying as artificially intelligent, these rabbits will not jump off the plane, nor will they clip through the barn. This is achieved by comparing the rabbits new coordinates with the coordinates they are allowed to be in. If a rabbit attempts to jump where they are not allowed, they will run into an “invisible wall” because their position coordinate will not update and they will fall straight back down to the ground.

When the user presses the ‘b’ key, a new rabbit generates using the same method as above.

When the user presses the ‘k’ key, a rabbit is removed at random.

Skyboxes

Skyboxes are generated using a cube map. They are then rendered first so that all objects in the scene render in front of them. As the user explores using the Flying Camera interface, they can see that the skybox behaves appropriately. However, when using the Circling Camera interface, the sky remains stationary. This decision was made because it was distracting and disorienting to have the sky move with the camera. To achieve this effect, the Circling camera does not update its ‘n’ or ‘v’ axes as it moves.

Day/Night Cycle

The main source of light, which we will call “the sun” follows a circular path. Each time the scene is rendered, the sun’s location is updated using the parametric equation for a sphere. With each render, the sun moves 1/10,000 of the way around the circle. When the sun dips below the horizon, the light source is turned off and the skybox changes to the nighttime sky. The seamless transition is achieved by having 2 skybox objects in memory, but only rendering the proper one according to the location of the sun

Code Organization

While imperfect, the code for this project is organized so that parts of code with different purposes are separated. The different parts of the project are split into directories named Common, Objects, Library, Textures, and Shaders. Each directory has its own “index.js” file, so that the main application isn’t littered with import statements or long path names.

Looking in the Objects directory, we can see how there exists a hierarchical relationship to the objects. For example, Bunny inherits from TLModel (textured and lit model), which inherits from BaseModel. This structure was chosen because it separates responsibilities and minimizes code duplication. For example, the BaseModel class strictly handles setting the model matrix and initializing properties necessary for a basic model. One step up, the TL model handles the texturing and lighting properties. Furthermore, the Bunny class handles the behavior specific to a rabbit.

Organizing the code using this model is much more manageable compared to my other assignments.

Handling Interaction

Most interaction is handled using a simple switch statement. All the keys that relate to the camera are delegated to the camera’s own interaction handler. The other features are handled directly within the app.js file.

The camera’s interaction handler uses a controller object that associates keys to functions within the camera class. When the camera’s handler is called, it takes the key and any modifiers to generate a "controller key." Then, if that controller key is defined in the controller object, the respective function is executed.

Picking

The algorithm used for picking in this project is… somewhat unorthodox. Instead of casting a ray, the program decides what was clicked on based on the color under the cursor. It takes the color under the cursor, divides each value by 255 (to get the percent color) and stores that as "test1". It then iterates through the list of rabbits. For each rabbit, it takes its ambient color and multiplies it with the ambient light color. This result is stored as "test2". Then, the Euclidean distance is calculated between the two vectors. If the Euclidean distance is less than 0.01, then that rabbit is removed.

Because of this algorithm, picking only works at night, when there is only ambient light. A better way to do this would be to render the scene to a frame buffer without any other light sources, and then perform the comparison that way so we can pick at any time of day. Or use the math-heavy method presented in class.

Work Allocation

I was the sole contributor to this project

Citations

All object textures not provided by the course were generated using architextures.org

Night Skybox texture provided by contributor "Fidaaaaa" at cleanpng.com

Bunny and cow models provided by David at https://www.cs.drexel.edu/~david/Classes/CS586/Models/