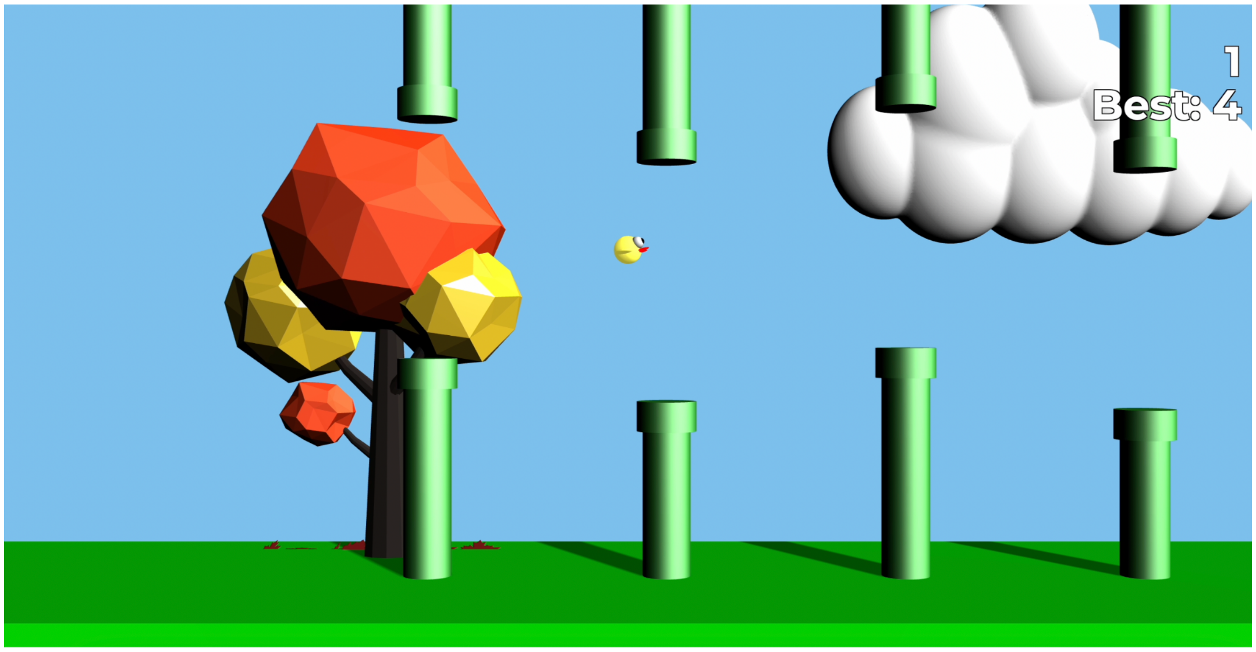
Tuesday, May 3, 2022

COS 426: Computer Graphics

Flappy Bird: Modern Graphics for a Classic Game

By Caroline di Vittorio and Julia Ruskin



Abstract

This project brings to life the classic iPhone game Flappy Bird by giving it a proper three-dimensional aesthetic. In this project, we reimplemented all of the features of the classic game, except that while the original game's graphics are retro, pixelated, and two-dimensional, our game uses three-dimensional meshes, lighting, and shadows for a sharper, more realistic look. Our project was successful, as we were able to successfully implement all of the core features of the game within a three-dimensional world. This paper details the approach, implementation, and design of our project.

1. Introduction

In May 2013, Dong Nguyen released a new game, Flappy Bird, to the iPhone App Store. By February 2014, the game topped the App Store charts, with over 50 million downloads worldwide.[1] The game's premise was simple: the user played as a small, pixelated bird trying to navigate through a series of green pipes. The user would tap the screen to make the bird jump up, and if the user didn't tap quickly enough, the bird fell down due to gravity. As the bird flew, green pipes moved onto the screen from the right side, and the bird would have to fly through the gap in between these vertical pipes. The game continued indefinitely until the user touched a pipe or fell onto the ground, at which point the game ended. The user's score was determined by the number of pipes they had successfully flown through before the game ended.

Though wildly popular, the original Flappy Bird used extremely simple, two-dimensional graphics. The bird, pipes, and background were all flat, two-dimensional, and pixelated, and there were no shadows or other advanced graphics. Moreover, the game's background was simple and remained constant throughout.

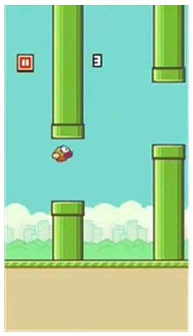


Figure 1: The bird flies through the gap between the green pipes in the original Flappy Bird, scoring a point.

In this project, we recreate Flappy Bird using modern, three-dimensional graphics. We set out to first reimplement the classic game, then bring it to life by representing the pipes and bird with three-dimensional meshes, adding lighting and shadows, and incorporating multiple background themes such that the game cycles through four different seasons. With our project, users can revisit the classic game (which is no longer available on the App Store), while also enjoying realistic, aesthetically pleasing, three-dimensional graphics.

Our approach relies on the Three.js library in JavaScript, from which we can easily load and create meshes and materials for the scene. We use the code provided by the course staff as a starting point, which enables us to dive right into developing our game without having to devote additional time to configuring our repository or setting up a server for testing. We began by reimplementing Flappy Bird with two-dimensional graphics first, and then gradually incorporated elements to create a three-dimensional game. This approach allowed us to create a minimum viable product quickly, ensuring that all the core features of the game were implemented before focusing on improving the graphics. With this approach, due to the short timeframe of the project, we were able to make the best use of our time by adding one feature at a time rather than potentially being too ambitious and starting features that we would not have time to finish.

2. Methodology

2.1. Initial Two-dimensional Implementation

Using the starter code, we created a rough draft of our game using two-dimensional graphics. We realized that it was considerably easier to first code the core features of the game in a two-dimensional environment, as it allowed us to understand how to handle collisions and where each position appeared on the screen. To do so, we used an orthographic camera and made each sprite from a two-dimensional image.

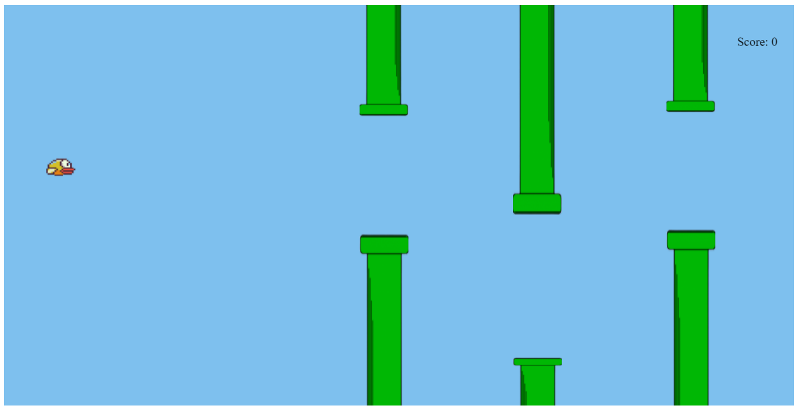


Figure 2: Initial Implementation of Core Features using Two-dimensional Graphics.

Our two-dimensional draft of the game implemented the core functional features of the game.

2.1.1. The movement of the bird: The bird remains at the same x-position on the screen throughout the game, and only changes its y-position to create the illusion of jumping. We use Tween.js to create the bird's animations. Initially, we considered simply incrementing the bird's y-position by a fixed amount every time the user pressed the space bar and decrementing it by a smaller fixed amount every frame to simulate the effects of gravity. However, we noticed that the resulting movements appeared unnatural: using Tween.js instead enabled us to make the bird's motion accelerate and decelerate, mirroring the way that objects behave under gravitational force in reality. Using an event listener, the program reacted to the user pressing the spacebar by starting a Tween.js animation in which the bird's y-position would increase by a fixed amount over a certain time period; the bird's upwards change in position would follow a quadratic curve, increasing rapidly at first and more slowly toward the end of the time period. Immediately after this hop ended, the program would check if any other hops were still being processed; if not, it would trigger a falling animation, causing the bird to accelerate towards the ground over a fixed time period. This way, the user was forced to repeatedly press the spacebar to keep the bird afloat.

2.1.2. The movement of the pipes: Each pipe is generated just to the right of the right edge of the screen, and its x-position is repeatedly decremented by a fixed amount on every frame. We chose to simply decrement the x-position rather than use Tween.js animations because we did not want the pipes to accelerate or decelerate, so decrementing their x position by a fixed amount was sufficient to achieve the desired visual effect. Every time the user scored a point, the pipes would move 1.1 times faster, making the game's difficulty gradually increase over time. For generating pipes, we chose to generate both the top and bottom pipes together, such that their total length summed to about 65% the height of the screen and each pipe covered at least 20% of the screen; this meant that the remaining 25% of the pipe length was randomly distributed between the two pipes. Finally, we chose to generate a new pipe once for every two units that the pipes moved rather than generating them every time a certain time period elapsed; this way, we ensured that despite any small lags, the pipes would be evenly distributed in space.

2.1.3. Scoring and collision detection: We implemented scoring and collision detection using bounding boxes. We counted a pipe towards the score whenever the bird's bounding box's minimum x-value exceeded a pipe's bounding box's maximum x-value. We kept track of the score using an HTML text element in the top right corner of the screen. We identified collisions between the bird and a pipe if the bird's bounding box intersected with the pipe's bounding box, and we identified collisions between the bird and the ground if the bird's y-position dropped below the bottom of the screen. If a collision occurred, we ended the game. Once the game ended, we stopped listening for the user pressing the spacebar and stopped updating the bird and the pipes, so the scene would stop moving. We also added an event listener for the 'x' key so that, after dying, the user could press the 'x' key to restart the game, reloading the scene and regenerating each of the sprites in their initial positions.

2.2. Moving to Three Dimensions

Once we had implemented the core functionality of our game, we were ready to switch to a three-dimensional environment, swapping the orthographic camera for a perspective camera. We also added a directional white light that was aimed towards the bird and the pipes, setting the `castShadows` property of the light and all of the objects in the scene to `true` so that shadows would appear. We then replaced the two-dimensional images that we had been using for our sprites with three-dimensional meshes.

2.2.1. Pipes: For the pipes, we recognized that a pipe was merely two cylinders: one large, narrow one for the pipe's body, and one small, wider one at the top of the pipe. Therefore, we used Three.js to create two cylinder geometries with fixed radii, one larger than the other, and used the same randomized scheme as before to determine the height of the longer pipe; we set the smaller pipe to be a fixed height, and set its position to be just above the top of the longer pipe. Finally, we equipped the pipes with a green phong material with shininess and specular reflection, ensuring that our pipes would reflect light and thus have a more three-dimensional look.

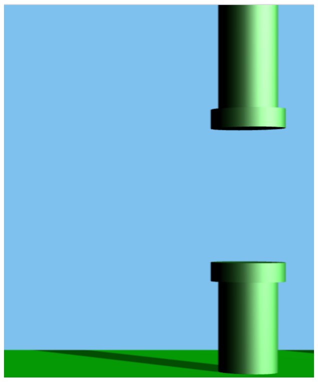


Figure 3: Three-dimensional pipes with phong reflectance and shadows

2.2.2. Bird: For the bird, we started with a mesh of a Flappy Bird that we found online,[2] but modified the mesh so that we could have the bird's wings flap up and down throughout gameplay. To accomplish this, we modified the mesh in Blender, using the Sculpting Mode functions, to create a second version of the mesh with the wings moved upward. We set the code to alternate between these two meshes every 20 frames. Moreover, to further simulate real motion, we made the bird rotate as it hopped: we added a Tween.js animation that coincided with the hopping animation, such that the bird rotated counterclockwise by 22.5 degrees (i.e. facing the sky) when it was jumping up, and rotated clockwise by 45 degrees (i.e. facing the ground), when the bird started falling.

To calculate intersections between the bird and the pipes, we continued to use bounding boxes. While it may have been more accurate to compute a bounding sphere for the bird rather than a bounding box, we found that there was not a significant loss in precision by not doing so, and Three.js made it far easier to use a bounding box than to compute a bounding sphere.

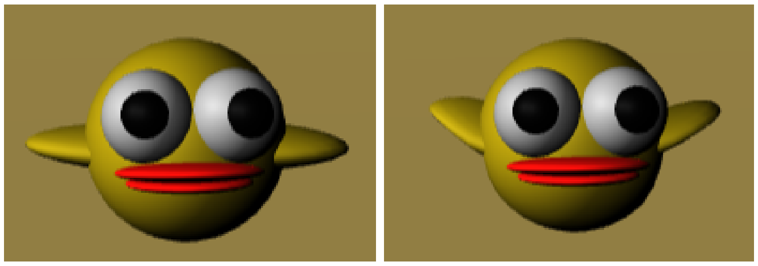


Figure 4: Our three-dimensional bird with its wings relaxed (left) and flapped upwards (right)

2.2.3. Floor: Finally, we added a floor and a bank beneath the floor at the bottom of the screen, both represented as plane meshes that we created using Three.js. To measure intersections between the bird and the floor, we simply checked if the bird's bounding box intersected with that of the floor, instead of checking that the bird did not drop below a certain y-position as we did in the two-dimensional version of the game.

2.3. Additional Game Mechanics

After migrating to a three-dimensional display, we decided to enhance the user's experience by adding text that welcomes the user to the game and provides instructions, text that informs the user when the game is over and tells them to press 'x' to start again, and text that displays the user's best score overall. We saved the user's best score in a cookie on their browser so that every time they access the game, they will be able to see their best score of all past attempts.

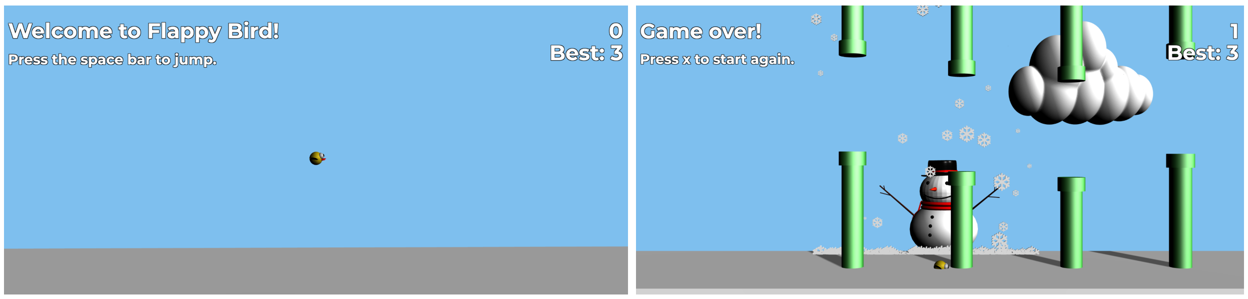


Figure 5: Example of Welcome (left) and Game-Over (right) Flow and Instructions Text

2.4. Scenery

Our final step was to add more exciting scenery in the background of the game. First, we added clouds, using a mesh we downloaded.[3] We rotated and scaled each cloud by a random amount in each direction, resulting in warped clouds that looked fairly different from one another. Each cloud was introduced at a random y-position in the sky, and it floated from the right side of the screen to the left side of the screen; clouds were introduced every time a given time interval passed.

Next, we decided to create a theme for each season. Every time the user first navigates to the page, a season is randomly selected; in subsequent games during the same session, the next chronological season would be selected (i.e. summer follows spring, fall follows summer, etc.).

2.4.1. Winter: We use the mesh of a snowman[4] to place in the background at the start of the game. We also use a mesh of a snowflake[5], which we used to repeatedly generate 10 snowflakes every time a new pipe was created. Each snowflake was generated at the top of the screen at a randomly selected x- and y-position, with a randomly selected size, and a randomly selected falling rate. The snowman and snowflakes all shift left alongside the pipes to give the illusion that the bird is moving forward.

2.4.2. Spring: We used the mesh of a flower that was provided with the starter code. We generate 10 flowers on the ground around each pipe, using randomly generated x and z position values to ensure that the flowers are spread throughout the ground; these flowers also move with the pipes in the x- direction as the scenery passes by.

2.4.3. Summer: We use the mesh of an ocean[6] (which we found online) and place these both behind the strip of land that the bird flies over; we also found meshes of a beach ball[7] and seashells[8] to randomly distribute along the ground. Again, each of these meshes moves along with the pipes in the x- direction.

2.4.4. Fall: We use the mesh of a tree[9] and a leaf[10]. We randomly generated trees at varying x-positions along the ground. Each tree would be accompanied by 10 leaves of randomly varying sizes, which fall to the ground as they appear on the screen.

3. Results

Overall, our project was a success, as we were able to implement a three-dimensional version of the classic Flappy Bird game, which runs any errors or issues that would make the experience less enjoyable. To test the game, we each played the game many times, ensuring that we played through each season's theme enough times to make sure that the game worked properly. We defined "working properly" as success in the following areas: handling collisions properly, moving without considerable lag, realistically simulating motion, and having no noticeable errors. To ensure that collisions were being handled properly, we tried running the bird into as many positions on the pipes as possible to make sure that the user did not lose prematurely or score points when it should have lost instead; we found this to be a success, as there were ultimately no instances when the program made a different decision about collisions than we would have ourselves. To ensure that lag was not an issue, we played the game multiple times with multiple other windows running in our browser, making sure that we did not notice any long, visible lags; we found that the program occasionally lagged, specifically on seasons with more complex scenery, but that these lags did not significantly affect our performance in the game. We made sure that our program realistically simulated motion by repeatedly jumping up and down with the bird and making sure that its rotations and movements seemed natural, which they ultimately, in our opinion, did.

4. Discussion

We believe that our approach of implementing the core functionality and subsequently improving the graphics was ultimately a success. Overall, we were able to apply a series of visually appealing effects to Flappy Bird that the original game lacked, resulting in what we believe to be a more enjoyable experience for the user.

One direction of follow-up work that could be taken is applying similar techniques to other popular, simple games. For example, the Dino Run game that Google displays when a user loses Internet connection could also benefit from a three-dimensional revamp. Another direction for follow-up work could be adding more complex mechanics to the Flappy Bird game. While we chose to stick with simply navigating the bird up and down for the sake of nostalgia, one could update our game to take advantage of its three-dimensionality; instead of only moving up and down, the bird might also move from left to right, navigating through pipes on multiple planes in three-dimensional space.

Overall, we learned a lot from this project. In addition to developing a much deeper understanding of Three.js, we learned more about how to make use of mesh representations of objects in practice, how to leverage the three-dimensionality of scenes that have perspective cameras, and how to cast lights and shadows.

5. Conclusion

In conclusion, in this project, we brought the Flappy Bird game to life by reimplementing the classic game with three-dimensional, modern graphics. In doing so, we successfully reimplemented the gameflow of the original Flappy Bird, as well as created four different detailed and new backgrounds for the game (themed after the four seasons). As next steps, we would love to take more advantage of the three-dimensional setting we've created, namely by expanding the game to take advantage of that space by, for example, having the bird move in the z-plane as well.

6. Academic Integrity

This represents our own work in accordance with University regulations.

/s/ Julia Ruskin '22 and Caroline di Vittorio '22

7. Works Cited

1. Kushner, David. "The Flight of the Birdman: Flappy Bird Creator Dong Nguyen Speaks Out." RollingStone. Last modified 2014. Accessed May 2, 2022. https://www.rollingstone.com/culture/culture-news/the-flight-of-the-birdman-flappy-bird-creator-dong-nguyen-speaks-out-112457/.

2. This work is based on "FlappyBird3d" (https://skfb.ly/oov7U) by sodasarbath2605 (https://sketchfab.com/sodasarbath2605) licensed under CC-BY-4.0 (http://creativecommons.org/licenses/by/4.0/)

3. This work is based on "ToonCloud" (https://sketchfab.com/3d-models/tooncloud-bba41664fb3e4f2b8979227b038046d0) by metin1 (https://sketchfab.com/metin1) licensed under CC-BY-4.0 (http://creativecommons.org/licenses/by/4.0/)

4. "Snowman 3D Free 3D Model." cgtrader. https://www.cgtrader.com/products/snowman-3d-bf5f7dfa-c726-4145-a727-4efe6202d4d8.

5. This work is based on "Snowflake" (https://sketchfab.com/3d-models/snowflake-5cb68fa2bd1a43eca4f0fc7f5c676a8d) by DoomShroom19 (https://sketchfab.com/DoomShroom19) licensed under CC-BY-4.0 (http://creativecommons.org/licenses/by/4.0/)

6. This work is based on "Ocean model" (https://sketchfab.com/3d-models/ocean-model-50f21b06c6e644e196b2ac828eda97dc) by conservationdude (https://sketchfab.com/conservationdude) licensed under CC-BY-4.0 (http://creativecommons.org/licenses/by/4.0/)

7. This work is based on "Beachball" (https://sketchfab.com/3d-models/beachball-ee18423fb5a54fa6bfb07094848feb70) by wildheart107 (https://sketchfab.com/wildheart107) licensed under CC-BY-4.0 (http://creativecommons.org/licenses/by/4.0/)

8. "3D Sea Shell 2 model." TurboSquid. https://www.turbosquid.com/3d-models/3d-shell-1562122.

9. This work is based on "a simple tree" (https://skfb.ly/6XXw7) by Negar (https://sketchfab.com/negar) licensed under CC-BY-NC-ND-4.0 (http://creativecommons.org/licenses/by-nc-nd/4.0/)

10. This work is based on "Canadian Leaf" (https://sketchfab.com/3d-models/canadian-leaf-b9198ec97d1848948c3be62b5b569085) by bimboxbob (https://sketchfab.com/bimboxbob) licensed under CC-BY-4.0 (http://creativecommons.org/licenses/by/4.0/)