

**CSED451 COMPUTER GRAPHICS**

**ASSIGNMENT 2**

2D HIERARICHAL DRAWING

SUBMITTED BY: ONG WEI HUA (49003139, ongweihua)  
 TAN WEI XUAN (49003140, tanweixuan)

TEAM NAME: SINGAPORE PEOPLE

DEPARTMENT: DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SUBMITTED TO: PROF LEE SEUNG YONG

SUBMISSION DATE: 29 MARCH 2019

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**2019**

# PROGAMMING ENVIRONMENT

The Programming Languages used are as follow:

* *FreeGlut 3.0.0-2*
* *GLEW 2.1.0*
* *GLM 0.9.9.1*
* *C++*
* *OpenGL & GLSL in Windows*

The Integrated Development used is

* *Microsoft Visual Studio 2007*

We also utilized the Source Control Platform, *GitHub*, and integrated it with Microsoft Visual Studio 2007 to allow us to be able to coordinate work between us and to allow us to easily track changes in our source code during our development process. The Repository can be accessed and view at the following link:

* *https://github.com/jermsinarocket/ComputerGraphics\_Assignment2*

# FUNCTIONALITY OF THE PROGRAM

The program that we implemented is a 2D Volleyball game that is based on the concept of the original “*Pikachu Beach Volleyball*” 2D side-scrolling game.

Enhancements have been made to the game that we have developed for our *Computer Graphics Assignment 1* the enhancements are as follow:

1. Flow of Electricity inside the ball (bounded by ball’s boundary
2. Horizontal Movements of Clouds
3. Repeated Changes to Clouds’ shape
4. Player’s ear and tail vibration (upon collision with ball)

These enhancements were implemented through the implementation of a hierarchical structure to each specific object.

# DESIGN AND IMPLEMENTATION

We assembled our games objects in a hierarchical way to compose new features to our objects out of multiple similar primitives. These new features positions are defined **relatively** to their parents. The way to implement this structure in OpenGL was to use *glPushMatrix()* and *glPopMatrix()*, which we have already being implemented in Assignment 1.

## Player’s Hierarchical Structure

To compose the player’s ear tail and to add motion to them, we added a hierarchical structure to the player object that we have implemented in *Assignment 1* is as shown below (Figure 1).

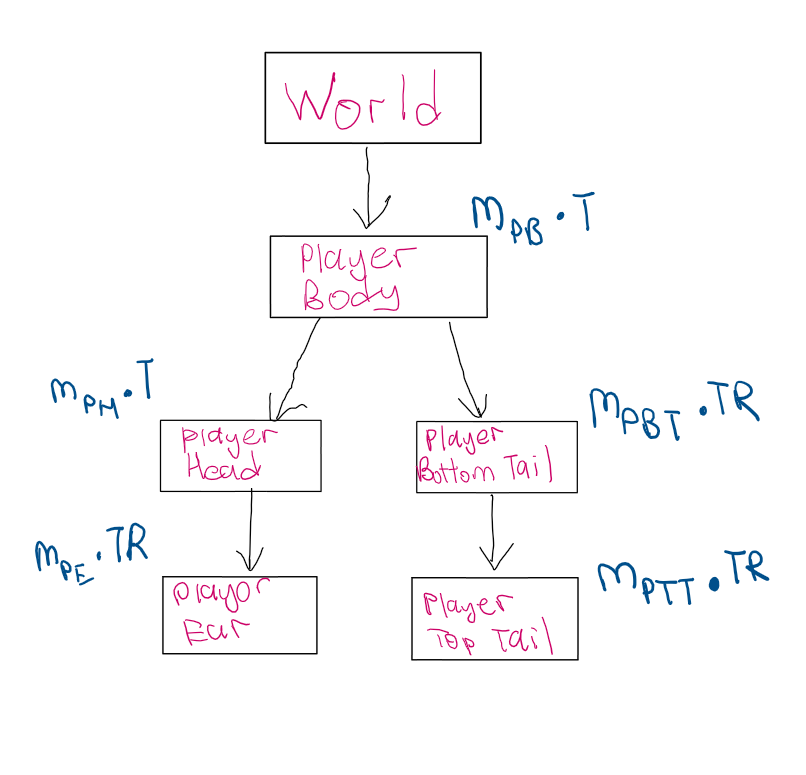


Figure 1 Scene Graph of Player

The **Player’s body** is the **top level** of the hierarchical structure, the **Head** and **Bottom Tail** are the **second level** and the **Ear** and **Top Tail** are the **third level** of the structure.

The implementation of the **Ears** and **Top Tail** are be documented below (Section 3.1.1 & 3.1.2).

### 3.1.3 Player’s Ear

The Player’s Ear is defined relatively to the Player’s Head on the third level of the hierarchical structure (Figure 2).

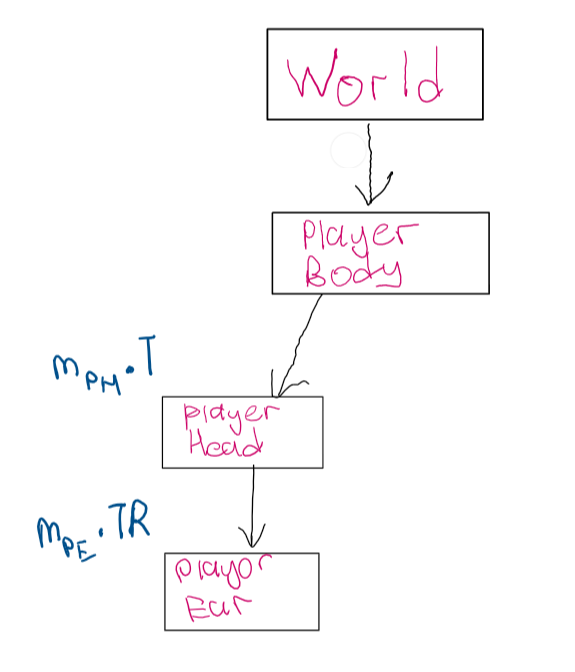


Figure 2 Scene Graph of Player’s Ear

The ear rotates about the point at which it is in contact with the head within a defined angle for a fixed period upon collision with the ball.

### 3.1.4 Player’s Tail

The Player’s Tail is split into two levels, the Bottom Tail (third level), relative to the Player’s body) and the Top Tail (fourth level), relative to the Bottom Tail. We implement the tail in the manner to add two separate motions to the Bottom and Top Tail respectively (Figure 3).

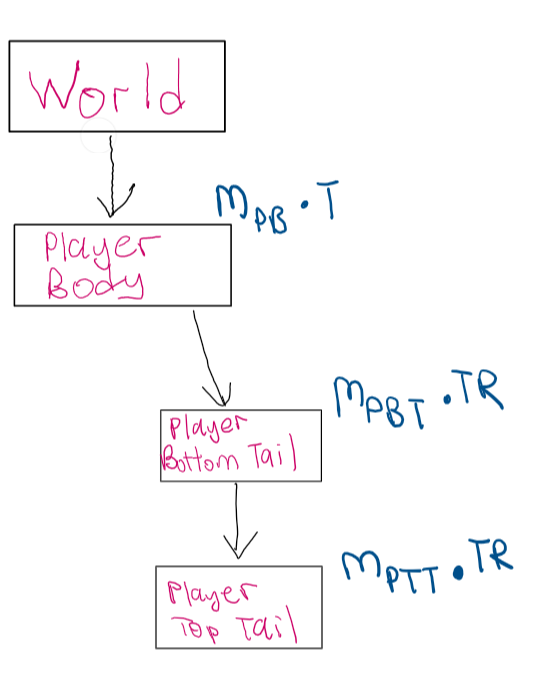


Figure 3 Scene Graph of Player’s Tail

The Bottom Tail rotates about the point at which it is in contact with the head, and the Top Tail does the same except that the point is with the Bottom Tail. Both parts of the tail rotates within a defined angle for fixed period upon collision with the ball.

## Cloud’s Hierarchical Structure

To compose the cloud’s left/right motion as well as the shape changes of the cloud, we implemented a hierarchical structure to it as shown below (Figure 4).

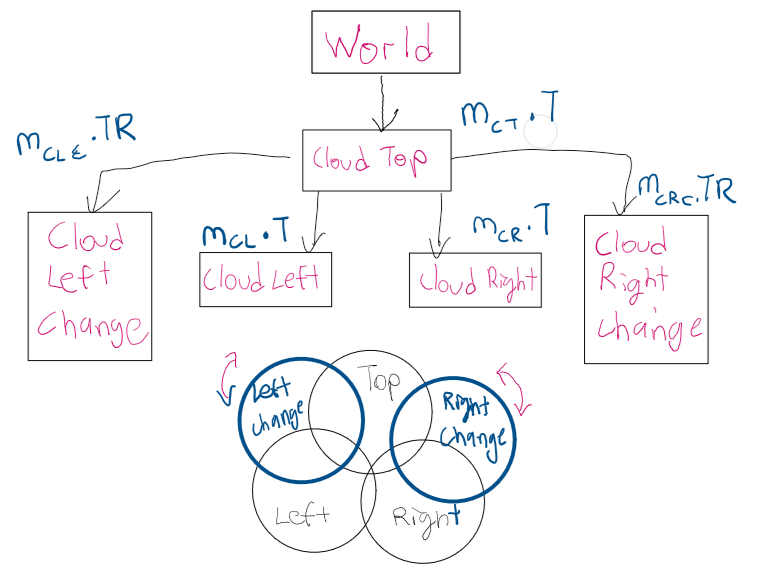


Figure 4 Scene Graph of Cloud

The **Cloud Top** is the **top level** of the hierarchical structure, the **Cloud Left, Cloud Right, Cloud Left Change and Cloud Right Change** are the **second level** of the structure and shape changes to the cloud is implemented through the Cloud Left Moving and Cloud Right Moving Objects. The implementation of the **Cloud Left/Right Change** is documented below (Section 3.2.1). Using this structure, translating the **Cloud Top** alone allow us to create the left and right movement for the entire cloud object.

### 3.2.1 Cloud Left/Right Change

The **Cloud Left** and **Right Change** objects are responsible for the shape changes to the cloud (rotation up and down). Both cloud objects rotates about the point at which it is in contact with the **Cloud Top** within a defined angle for an infinite period (Figure 5).

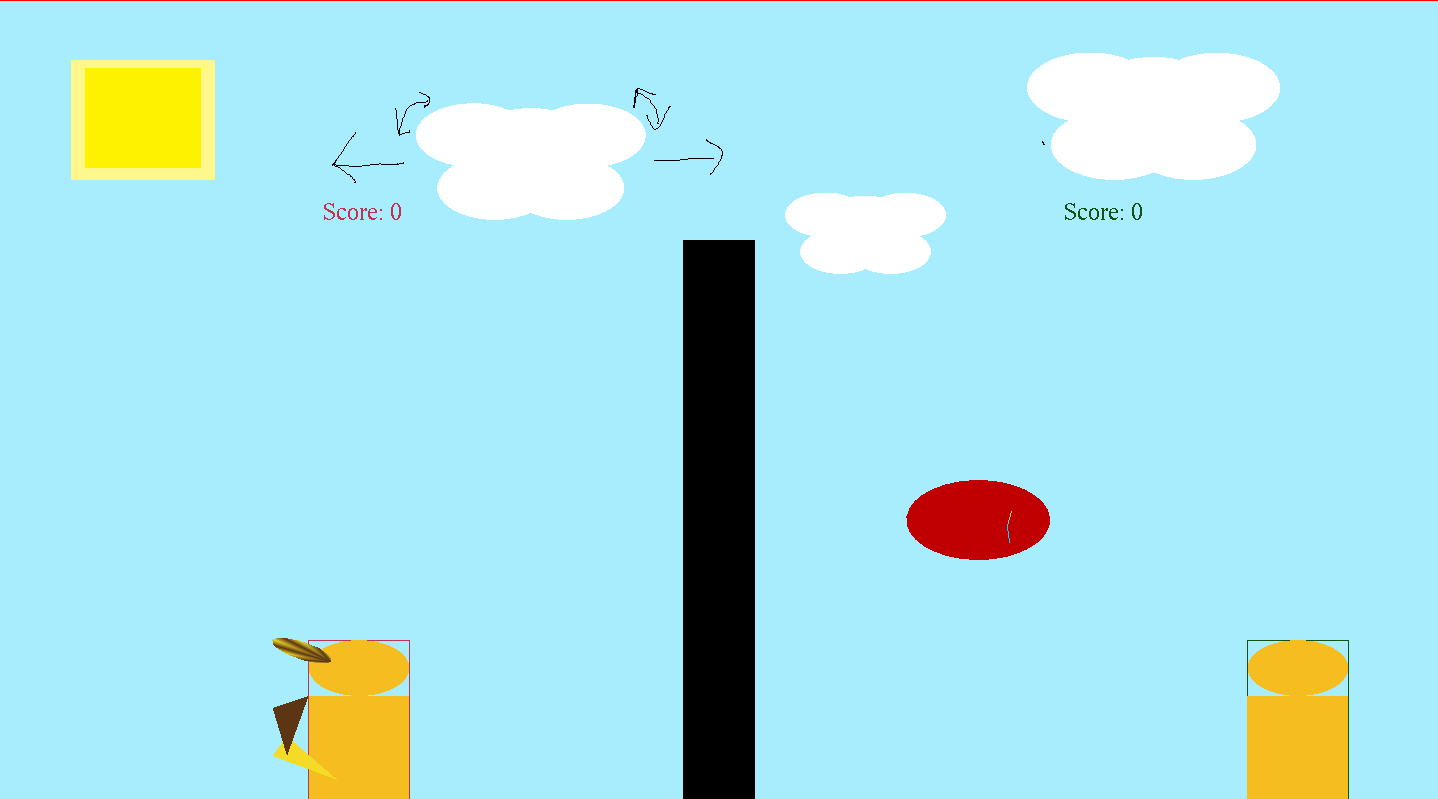


Figure 5 In-Game Cloud Motion

## Ball’s Hierarchical Structure

To compose the flow of electricity within the cloud, we implemented a hierarchical structure to the previous ball object that we have implemented in *Assignment 1* as shown below (Figure 6).

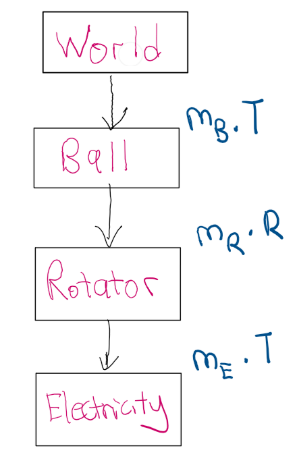


Figure 6 Scene Graph of Ball

The **Ball** is the **top level** of the hierarchical structure, the **Rotator** is second level of the structure and it is in change of the rotation of the electricity within the ball. The **Electricity** is the **third level** of the hierarchical structure. Using this structure, the Electricity will be bounded within the ball’s by default.

### 3.3.1 Rotator & Electricity

The **Rotator** is a hidden triangle object to control the rotation of the electricity within the ball. The Rotator rotates about the centre point of the ball for an infinite period.

The **Electricity** is a series of randomized line segments relative to the bottom left and right coordinates of the Rotator, meaning that the length *(x)* and height *(y)* of the line segments (Components that draw the Electricity) are constantly changing.

The implementation of the Rotator & Electricity was designed in accordance to the sample GIF that was being provided to us (Figure 7).

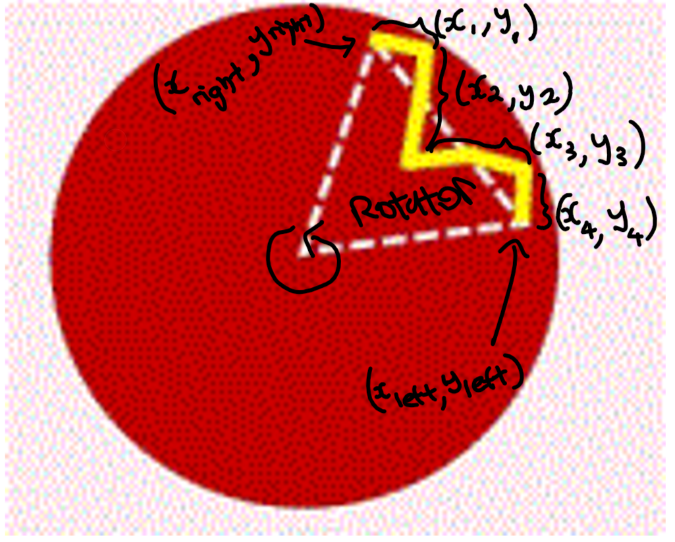


Figure 7 Ball Model

## Restarting the Game

The function that restarts the game is rather simple to implement. Pressing the “r” key will reset to the game to its original state. This function is implemented by resetting all the objects to its initial coordinates. The initial coordinates of all the objects are **defined** and set to be **static** so that it will be **unchanged throughout the execution of the problem**.

# BRIEF EXPLANATION OF THE PROGRAM

## Running the Program

There are two ways to run our program, through Microsoft Visual Studio or directly running the executable file:

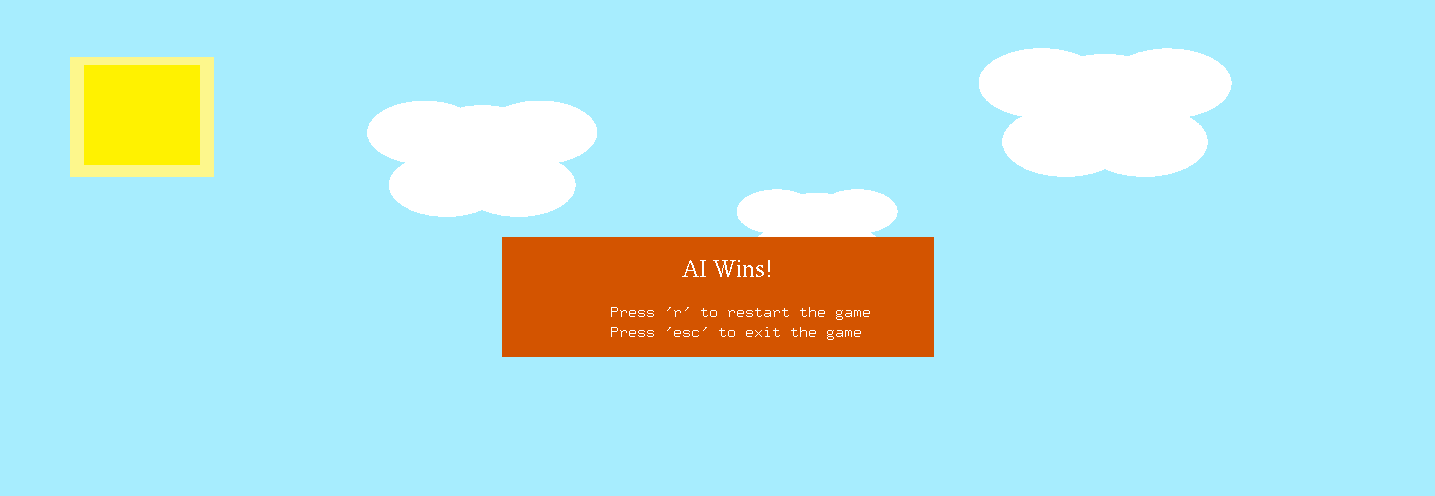
1. To run the program through Microsoft Visual Studio, launch the ***Assignment\_2.sln*** file located in the ***Assignment\_2* *folder***. After Microsoft Visual Studio has been launched and the solution has been loaded, **build the Solution** *(Ctrl + Shift + B)* and then run the program by pressing **F5**
2. You can launch the program directly by running the executable file ***Assigmnent\_2.exe*** located within the ***bin folder*** *(Assignment\_2\bin)*

## Playing the Game

No additional components have been added to the aspect of gameplay and thus the game structure is like *Assignment 1*. Details on the gameplay can be found by referring to our *Assignment 1*’s report.

## Restarting the Game w/ Examples

At the end of the game (whichever player wins), the player will need to press the ‘r’ key to restart the game. This will bring back the game to its initial state.





# Additional Features

We have implemented several additional features to our program and they are as follow:

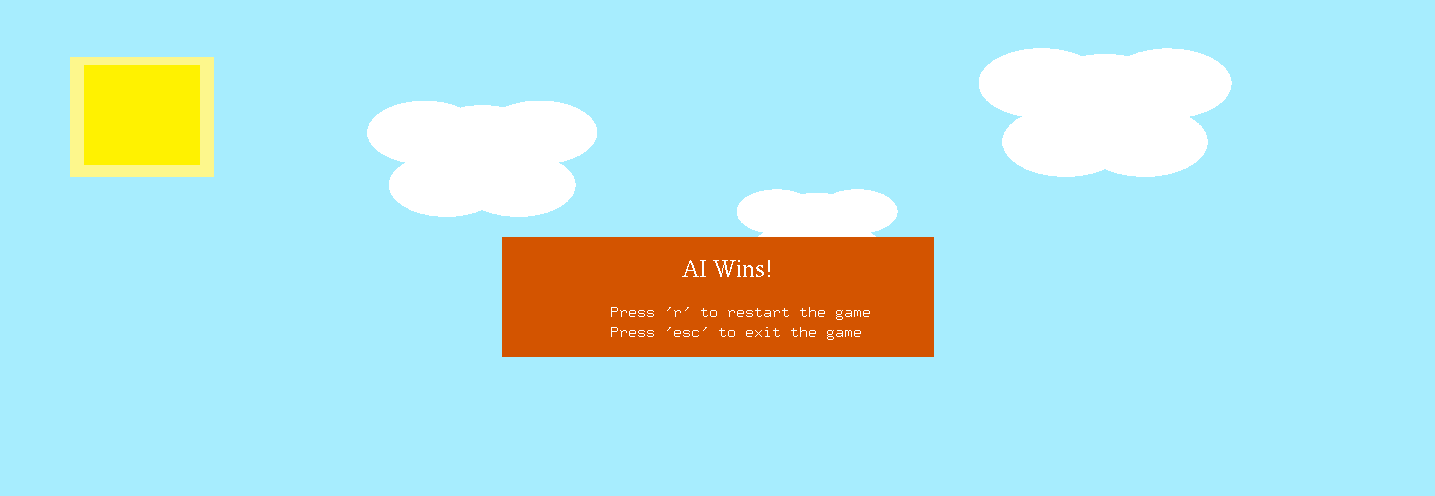
1. **Game Sounds**

* We have implemented Game Sounds at different portions of the game
  + 1st Portion: Running the Game will play **the Game Theme Song**.
  + 2nd Portion: Clicking the “Start” Button will trigger the **Start Sound**.
  + 3rd Portion: When the ball collides with points within its defined collision region, a **Bouncing Sound** will be played.
  + 4th Portion: When the Player/AI scores a point, a **Cheering Sound** will be played.
  + 5th Portion: When the Player/AI wins the game, the **Ending Theme Song** will be played

1. **Exiting the Game**

* User will be able to exit the game straight by pressing the ***“esc***” key.

1. **Switching between Fixed/Partial Screen Modes**

* In addition to showing a partial screen mode, we implemented a function whereby users can toggle between the fixed and partial screen modes by pressing the “z” key. 

# PROBLEMS FACED AND SOLUTIONS

The implementation of the hierarchical model to the Player, Cloud and Ball was rather simple for us as we have already utilized a similar structure to create out objects in our *Assignment 1.* However, the two main difficulties that we encountered were the drawing of the Player’s ear and the flow of electricity within the ball.

Secondly, the Pikachu ear’s shape was hard to draw on 2D plane due to its nature of its shape. It was neither a circle or square but rather it was an oval shape. After looking deeper into how we previously drew a circle, the shape of the ball, we realized that unlike a circle which have a fixed radius, we could draw an oval by having a separate horizontal and vertical axis. By doing so and applying a technique similar how we drew the shape of the ball, we were able to create the shape of the player’s ear.

Lastly, the main difficulty that we faced was implementing the flow of electricity within the ball. Rotating the electricity within the ball was rather easy for us as all we had to do to create an infinite loop that translates the rotator back to the origin, rotate it and translate it back to its original position. The main difficulty that we faced for this portion of our assignment was in changing the shape of the electricity repeatedly to show its flow. Initially, we thought that we had to manually create multiple line segments with varying [x, y] values to join the bottom left and right segments of the rotator. However, we were faced with one big issue – changing the shape and length of the line segments on each render. After much thinking, we realized we could create a loop to create N line segments with randomized [x, y] values on each render. To prevent the line from flowing out of the boundaries of the ball, we constrained the x-value to be within the x-axis range of the bottom left/right of the rotator and the y-value to be within the bottom y-axis value of the ball.

# IMPROVEMENT TO PROGRAM

There are still numerous improvements that can be made to our program.

Firstly, the clouds are currently moving from left to right. Improvements can be made if we infinitely generate new clouds that can move in and out from the game screen.

Next, the current program is implemented on a 2D platform. We could further enhance it by changing our entire gameplay to 3D.

# CONCLUSION

In summary, through this assignment, we have gained a better understanding about the concepts of using hierarchical modelling to allow us to create and apply motion through translations and rotations to objects in a much more efficient and easier way. The most important aspect aw that we were able to apply the concepts that were taught during class to our assignment and gained an even better understanding them. All in all, this assignment has helped further understand the usefulness and implementation of hierarchical modeling in OpenGL.