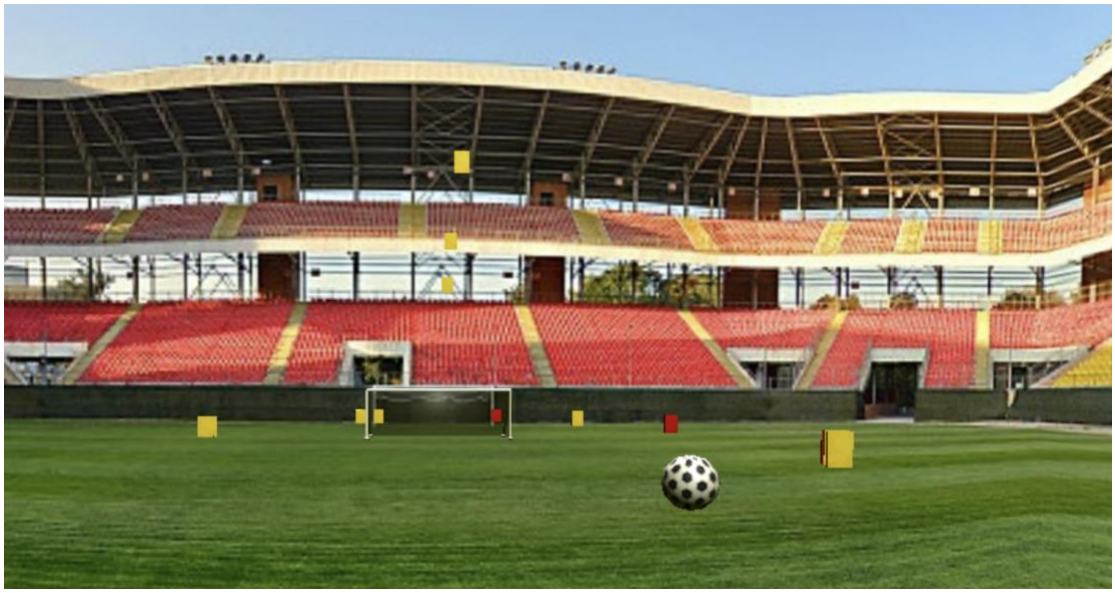
Reichman University: Computer Graphics Course 2024
Instructor: Prof. Ariel Shamir
Submission due: 23rd of June 2023 (23:59)

Exercise 6 – Creating with Three.js



Overview

In this exercise, you will use the elements from the previous exercise to implement a small game in Three.js!

The game will have you control a football, trying to score a goal, using the arrow keys, and avoiding red and yellow cards.

Reichman University: Computer Graphics Course 2024
Instructor: Prof. Ariel Shamir
Submission due: 23rd of June 2023 (23:59)

Modeling

You should be using your goal and ball from the previous exercise for your exercise. We will not grade it again - meaning if you've had a points deduction in the prior exercise, we will not penalize it again.

You must change the opacity of the goal's nets, making them half-transparent (the exact amount is your choice). Important: even if you decide to change the nets in any way (such as textures), they should still be half-transparent.

You will have to add at least three yellow cards and three red cards - we suggest using shapes like boxes, but you can simply use planes (it does not matter which shape you choose).

You will also have to use textures to make the cards look lifelike (textures will be added).

Gameplay Interaction

The rules for the game will be:

- The ball always starts in front of the open side of the goal and ends inside\passing the goal.
- There are (at least) 3 different routes to the goal, each using their own Bezier curve.
- The ball moves along one of the Bezier curves while spinning around itself.
- The ball moves from one route to another using the arrow keys (Left/Right)
- The ball needs to avoid the cards in each route. Each card contributes **negatively** to the Fair Play score according to the formula: $FairPlay = 100 \times 2^{(-\frac{numYellowCards+10\times numRedCards}{10})}$. Note that the score is always positive.
- When the ball gets into the goal, there's a prompt that notifies the player what is the Fair Play score.
- We recommend that the routes start at (0, 0, 100) and finish at (0, 0, 0).

Viewing

The player's perspective should have a good view of all the routes so the player will be able to switch and avoid all the cards. The camera should be following the ball's movement towards the goal, but only on a certain axis - the camera shouldn't be affected by movement from route to route.

Reichman University: Computer Graphics Course 2024
Instructor: Prof. Ariel Shamir
Submission due: 23rd of June 2023 (23:59)

Materials

As we mentioned before, you will have to use **MeshPhongMaterial** for your objects. Like last time, we are giving you some artistic freedom and you may color all the objects in any color you'd like **other than the ball**, which you are required to use a specific texture.

Lighting

You are required to add at least three light sources:

- **2 Directional Lights** above the start and the end of the routes.
- **Ambient Light**.

You may add more lighting sources other than those, for example, a Spot Light.

Bezier Curves

Creating the routes should be simple - use the Quadratic Bezier Curve to simply draw a Bezier curve for the ball to travel on. Use the Curve documentation to understand how to actually work with it.4

If you used our recommended start and end points, we recommend using these control points for your routes, but you may create your curve or even use the Cubic Bezier Curve.

Right Winger Route: (50, 0, 50)
Center Forward Route: (0, 50, 50)
Left Winger Route: (-50, 0, 50)

Note: You may notice when traveling between curves that the ball will not "feel" in the same place. This is because the delta between each point in the curve may not be constant.

How to move the ball

You will need to make sure that in each frame, the movement of the ball will be direct ed to the new position on the curve.
To know where on the curve we should be, we can use the *.getPoint* method.
Hint: You have both the current world position of the ball (using the *position* attribute) and the world position on the curve. To create the translation matrix, some basic arithmetic calculations may be required.

We recommend splitting all the curves in to a constant number of increments between 3,000 to 10,000 , with each animation frame moving a single increment.
Note: Since we are working by frames and not by time, each computer may react differently speed-wise. You may use some coding tricks to bypass that, but it is not required (see bonus)

Reichman University: Computer Graphics Course 2024
Instructor: Prof. Ariel Shamir
Submission due: 23rd of June 2023 (23:59)

Placing the cards

We require **at least** 2 cards in each curve.

We recommend implementing it in such a way that there's a data structure that holds:

- The curve it belongs to (reference)
- The *t* value on the curve it appears at
- The Object3D of the card

And of course, a list that holds all the cards.
Hint: Ordered list by *t* value may prove handy later on.

Collision Interaction

You may wonder how we can detect when two objects collide with each other. This usually takes some kind of physics or ray-casting techniques to resolve, but we will use a much simpler solution.

Since we know at each time what is the current *t* value and which route the ball is at, we can check only whenever we are expecting a collision - meaning there's a card with a similar *t* value as the ball (remember that the ball has volume, so a collision can "occur" in multiple places. Your solution should take your ball's volume into account). If there's a card with a similar *t* value **and** route as the ball, we can consider it a collision. Therefore, we can use the *.visible* attribute to make it *false*, making it disappear, and incrementing the internal score value.

Bonus Features

We are adding a list of suggestions to add to your game, each has a different point value. **If you choose to implement one of the bonuses, add a README that states which bonus you've implemented. Note that the maximal bonus is 10 points.**

- Add a flag of your favorite club \ country (1 point)
- Add a modeled trophy (1 point)
- Make the ball spin around itself while moving (1 point)
- Add more scenery (1 points)

- Make your game time-dependent instead of framerate-dependent. (2 points)
- Add collectible cards with "VAR" written on them that increase your Fair Play score instead of decreasing it (3 points)
- Any creative idea that you have (up to 5 points)

Recommended Milestones

While it may look simple, this exercise might be overwhelming. We recommend completing it by the milestones, and not doing everything at once:

1. Start by importing your goal from the previous exercise into the scene.

2. Create a sphere for the ball and add the texture for it.
3. Add lighting to the scene, to make sure everything is rendered properly.
4. Time to add Bezier curves. Start with a single curve. We recommend at first rendering the curves for visualizing - eventually you will have to remove it, but this is a good helper tool.
5. Now, create a simple animation of the ball traveling along the Bezier curve.
6. After you have a traveling ball into the goal, time to make sure you have a good view of it - make sure that the camera follows the ball in both position and view.
7. Add the rest of the curves and build the traveling between curves using the keyboard. You may implement it either in a circle (after reaching to the rightmost curve return to the leftmost) or not.
   Hint: Create a list with all the curves, and another variable that holds the index of the current curve.
8. It's time to add the cards. As always, start with a single card and add it to the scene on a specific increment of the curve.
   Hint: You might find it easier to create a class as mentioned before, but it's not a must.
9. Test the "collection" of the single card - use the methods described in the "Collision Interaction" part.
10. Time to multiply it - create a for loop that creates cards and add them to a list.
    Hint: It is recommended to create the card list ordered by *t* value. This way, we can always check only for the next closest card (and if we "missed" it, move to the next one)
11. Add a counter for each collected card, and fire a prompt at the end of the curve.

Tips

- Work smart and organized, and add comments to your code. It is recommended to use the code block sections we wrote for you.
- Test all the time! The main advantage of WebGL is the ability to test fast and see the errors and bugs visually.
- Everything builds upon each other - don't move on to another feature before you are finished with another.
- Use version control. You might have a great version but then change something, and it will ruin everything by mistake - better have backups.

Grading

- Basic environment (ball, goal) - 10 points
- Implement Bezier Curves - 10 points
- Ball trailing Bezier curve animation - 10 points
- Crossing between curves using keys - 20 points
- Adding cards along the curves - 20 points
- Collision detection of ball and cards - 20 points
- Scorekeeping and score prompt - 10 points

- Bonus – up to 10 points

Submission

- Submission is in pairs.
  - Zip file should include only a single script: hw6.js
  - **A short README document and any additional files if you implement a bonus.**
- Zip file should be submitted to Moodle.
  - Name it: Ex##_FirstName1_Surname1_ID1_FirstName2_Surname2_ID2.zip
- Before submission, be sure to check for updates on Moodle and Piazza.
- After submission, download the exercise again, replace the given hw6.js with your hw6.js, and run again to verify that your solution works.
- **There will be a point reduction of up to 10 points for submitting incorrectly.**