CS606 Computer Graphics

Term 2 2021-22

Programming Assignment 4

Virtual Reality & Animation

Contributors:

ADRIJ SHARMA - IMT2019004

SAMAKSH DHINGRA - IMT2019075

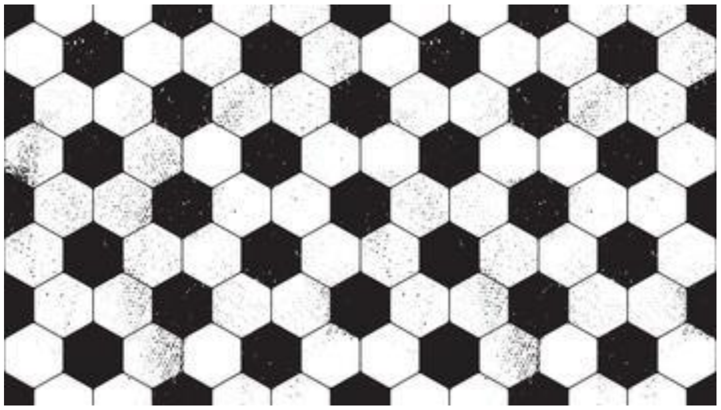
ABHAY ARAVINDA - MT2021002

ANSWERS

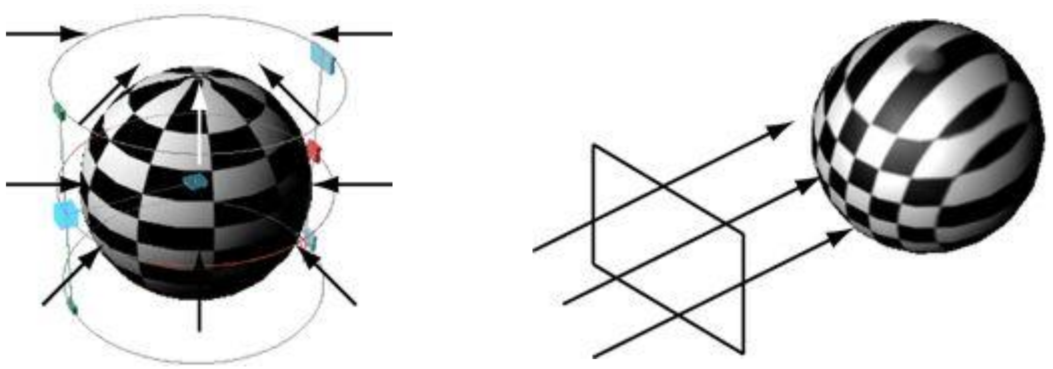
1. ***What kind of distortions do you notice with the texture maps you have used? What would be your approach to correcting them?***
   We manually implemented texture mapping for Football and Rectangular football field. While implementing texture mapping on these objects we faced the following issues:-

   Football

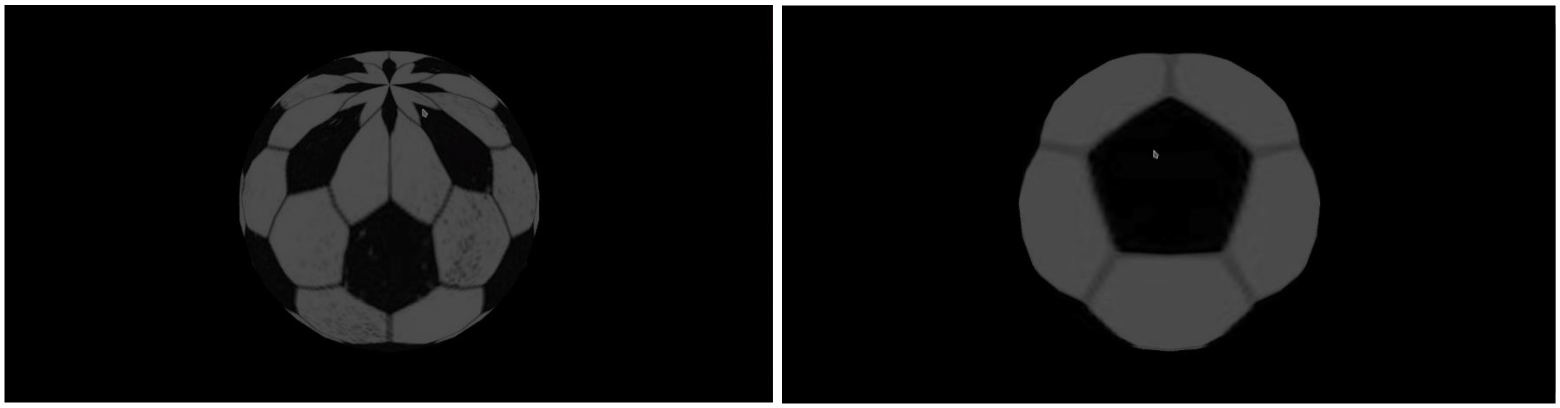
   We first mapped the spherical object (from Part 1), with the following rectangular football map image:



   But we saw distortions in the football like overlapping black pentagons at top and bottom part of the sphere with planar and cylindrical mapping which can be seen as follows.



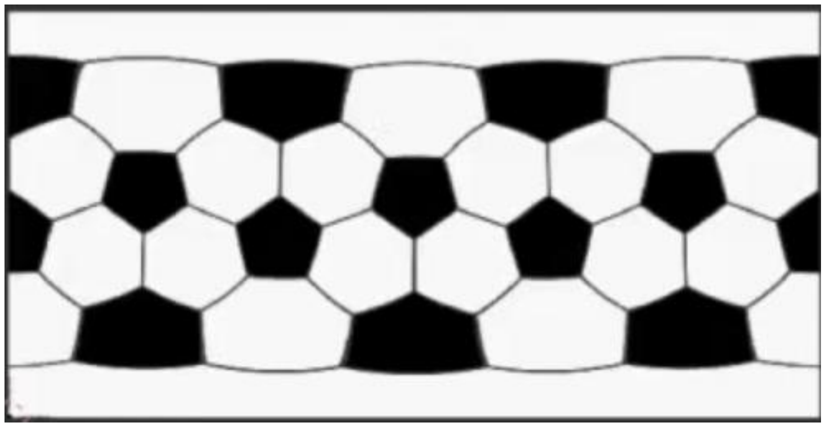
<u>Reference Images</u>



<u>Distortion and correction in our project</u>

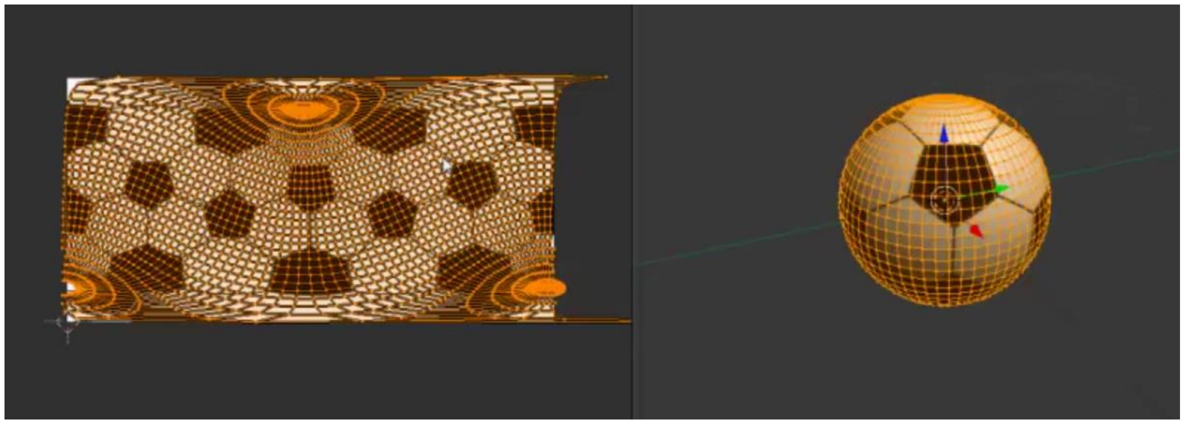
Approach to correct them:

We used correct spherical texture map image of the football like following:



The main feature of this correct texture map is that it is a surface parameterized spherical map of the football. So, it can be now used effectively by the mapping techniques.

This texture is mapped to the spherical ball using technique of UV Mapping (Customized):



This technique is already implemented in three.js libraries so we just used direct functions to implement this functionality by just loading the correct spherical texture map image of the football.

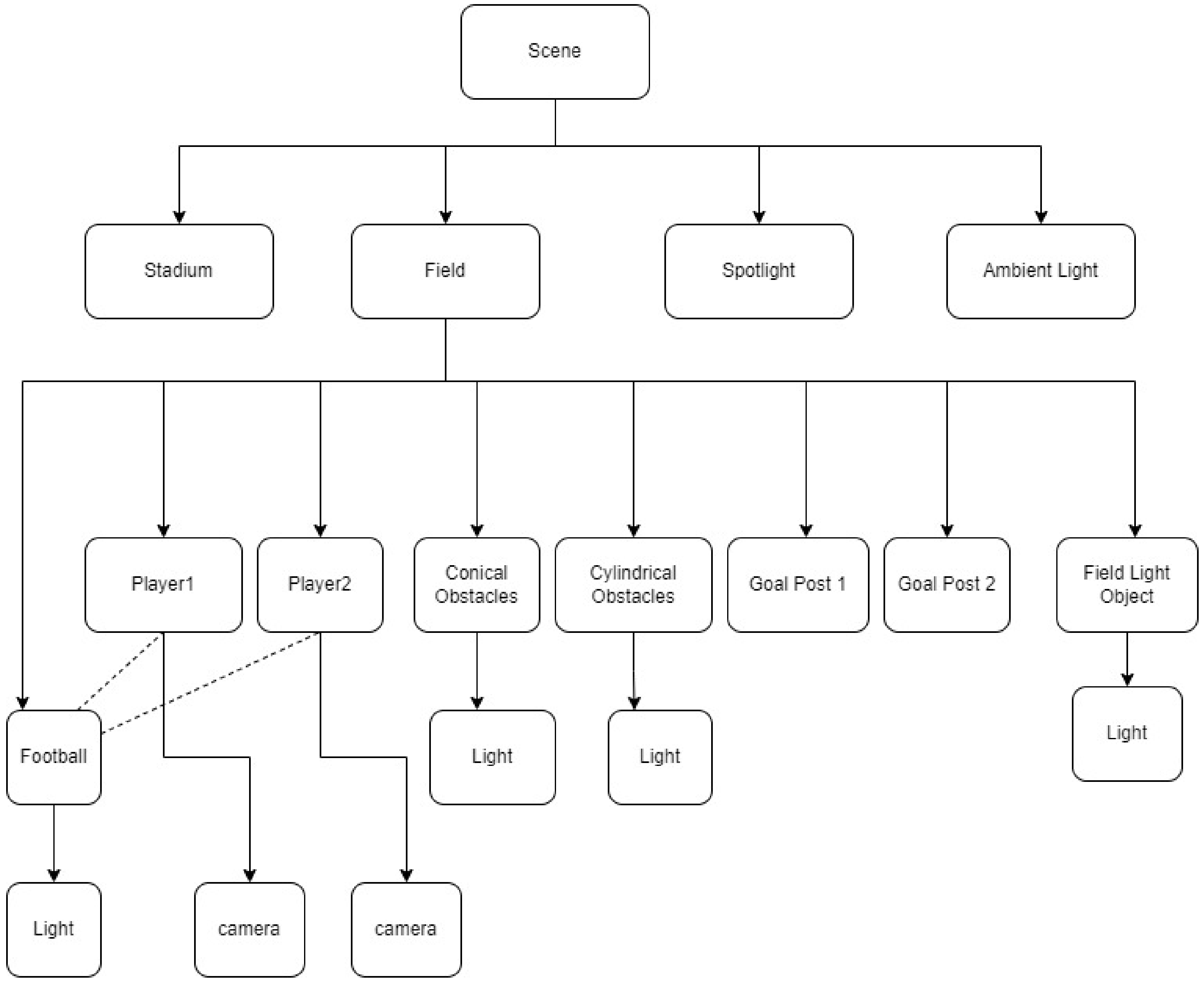
Rectangular Football field

We implemented texture mapping on football field using planar mapping with the football field texture. In this we faced following issues:

- Blurr due to small texture image and large field object: We fixed this by using a large image.
- More Stretch in one direction because of different aspect ratio of field and image: We fixed this by adjusting aspect ratio of the field object
- Repeated texture images on field due to small image: This was fixed by using different function in three.js that disabled texture repeat.

For the players and the rest of the objects like stadium, goalpost, obstacles, light poles, etc. the default texture coordinates are used, so we did not face any issue with them as they were already made models with their material properties (including the textures) and we just imported them.

2. ***Provide a brief writeup (1-2 pages) on the design you used for:***
   a. ***Scene graph organization***



Scene graph plays an important role in the complete scene, facilitating easier relative transformations for the objects. The above figure depicts the scene graph for our project. Our scene consists of a 3D mesh consisting of stadium, field and spotlight of the two players in play. The field consists of 2 players (with separate cameras on each player), a football, conical obstacles (with light), cylindrical obstacles (with light), 2 goal posts, and the field light objects. The football, the conical and cylindrical obstacles, and the field light objects have their separate lights independent of each other's light. We have an Ambient light of pure white color (0xffffff) and intensity of 0.2 present in the scene.

b. ***How the position and orientation of lights and the avatar's camera are calculated.***

Ambient Light – This light does not have any position or orientation. It has been used for giving minimal light so that the stadium does not appear dark. It has been modelled in the scene. Intensity of this light is 0.2 and colour is white.

Point Lights – A point light has been positioned inside the football and it moves along with the football. It has been modelled in the scene graph to be inside the football. The light being inside the ball only illuminates inside surface of the ball. To enlighten the ball, we temporarily change the material of the ball to a material which is independent of lights in the scene. The material of the ball has been stored in a private variable. When the light is on – standard material (glowing material), when the light is off – normal material.
Each obstacle too has its own point light which is positioned at the same position as the obstacle and remains fixed in the scene throughout the game play.

Spot Lights – There are 6 fixed stadium lights which are oriented towards the field. Out of these 6 lights, 4 lights are positioned on 4 corners of the field and 2 of them are positioned in the center of the 2 lengthy sides of the field. Each of these 6 lights face the field and are oriented towards the field. Each player also has his own spot light which remains on top of the player all the time and moves along with the player. This light can be toggled on and off. Each time, just before the render updates, the light's target is set to player's position.

Cameras - There are two sorts of cameras available: global camera and first-person camera. Both global and first-person cameras are implemented as perspective cameras. The camera in first-person view is designed to operate like the Player's eyes, with the same position as the head and is supposed to move with the player. The player has no connection to the World Camera. The first-person camera presented the most difficulty, as it was tough to come up with a reasonable formula for getting the camera to look at the correct vector. This has been implemented using the trackball controls class of threeJS. We calculate the camera's position in relation to the player, as well as the point at which it is supposed to look. We pass these settings to the camera and therefore avoid any problems. The positions were established in relation to the player, and the player's final position was computed using matrices.

The global camera is able to look at the entire scene and everything that is going around. It can zoom, pan and rotate in the entire scene.

c. ***Computation of animation including collision detection/avoidance.***

Collision:
- ➢ Collision detection has been implemented through bounding boxes.
- ➢ If the ball collides with an obstacle, then the direction vector of the ball is negated and a randomness factor is added. The ball rebounds in a random opposite direction.
- ➢ Both the players and the ball have a bounding box. If these bounding boxes intersect then the ball comes in the control of the player.
- ➢ If a player collides with an obstacle –
  - If he was walking, then he will be pushed back. We compute the forward vector of the player and move him back in a proportionate amount.

    - If he was running then he will be pushed back and ultimately fall down on the ground. If the player had the ball with him, then he will lose the control of the ball.
- If two players collide with each other –
    - If a player collides with the other and none of them had the ball in his control, then the moving player will be knocked back.
    - If two players collide and one of them has the ball in control, then the player who had the ball in control will fall down and also lose the ball control.
- If the ball collides with the goal, then a goal is detected and the score is updated appropriately.

Animation:

- Entire player animation is done with .fbx object.
- During falling and rising up again and during kicking, the player cannot move. Translation is disables during these activities.
- Translation of a player is done by changing the player position.
- In carrying mode, ball modelled as a child of the player and moves with the ball balanced between his head and his back.
- In dribbling, the football can also be observed as rotating.
- Football, has a shoot mode. It has a direction and a distance that it can travel. The direction may change if it collides in the middle.

CODE SPECIFICS & APPROACH

- Most of the implementation details have been explained through the answers to questions.
- We implemented assignment 3 in WebGL and assignment 4 in ThreeJS, therefore, it is practically not feasible to use the code of assignment 3 in assignment 4. But, concepts of A3 have helped to accomplish A4.
- A player carrying the ball in his hands does not make sense while playing football, therefore, we have implemented the animation where a player is able to balance the ball between his head and his back.
- Instead of kicking the ball always in the direction of the goal, we instead have implemented the kick of a player in a way that the ball always gets kicked in the forward direction.
- Boundary conditions: If the world coordinates of the ball cross a threshold, and the ball goes out of scene, then the position of the 2 players gets reset to the initial position when the game started and the game resumes with the same score.

REFERENCES

1. ***Documentations:***
    - https://threejs.org/docs/
    - https://threejs.org/docs/#api/en/lights/AmbientLight
    - https://threejs.org/docs/#api/en/textures/Texture

- https://threejs.org/docs/#manual/en/introduction/Animation-system
- https://cs.wellesley.edu/~cs307/threejs/mrdoob-three.js-d3cb4e7/docs/
- https://github.com/mrdoob/three.js/

2. ***Videos***
   - https://www.youtube.com/watch?v=NNpMem-MiuI
   - https://www.youtube.com/watch?v=pUgWfqWZWmM
   - https://www.youtube.com/watch?v=2IWjCvTCeNE

3. ***Repositories***
   - https://github.com/JieYang13/WebGIS-football
   - https://github.com/Artwewe/ThreeJs-Car-Football
   - https://github.com/kamilkulig/three-js-game
   - https://github.com/matthias-schuetz/THREE-BasicThirdPersonGame