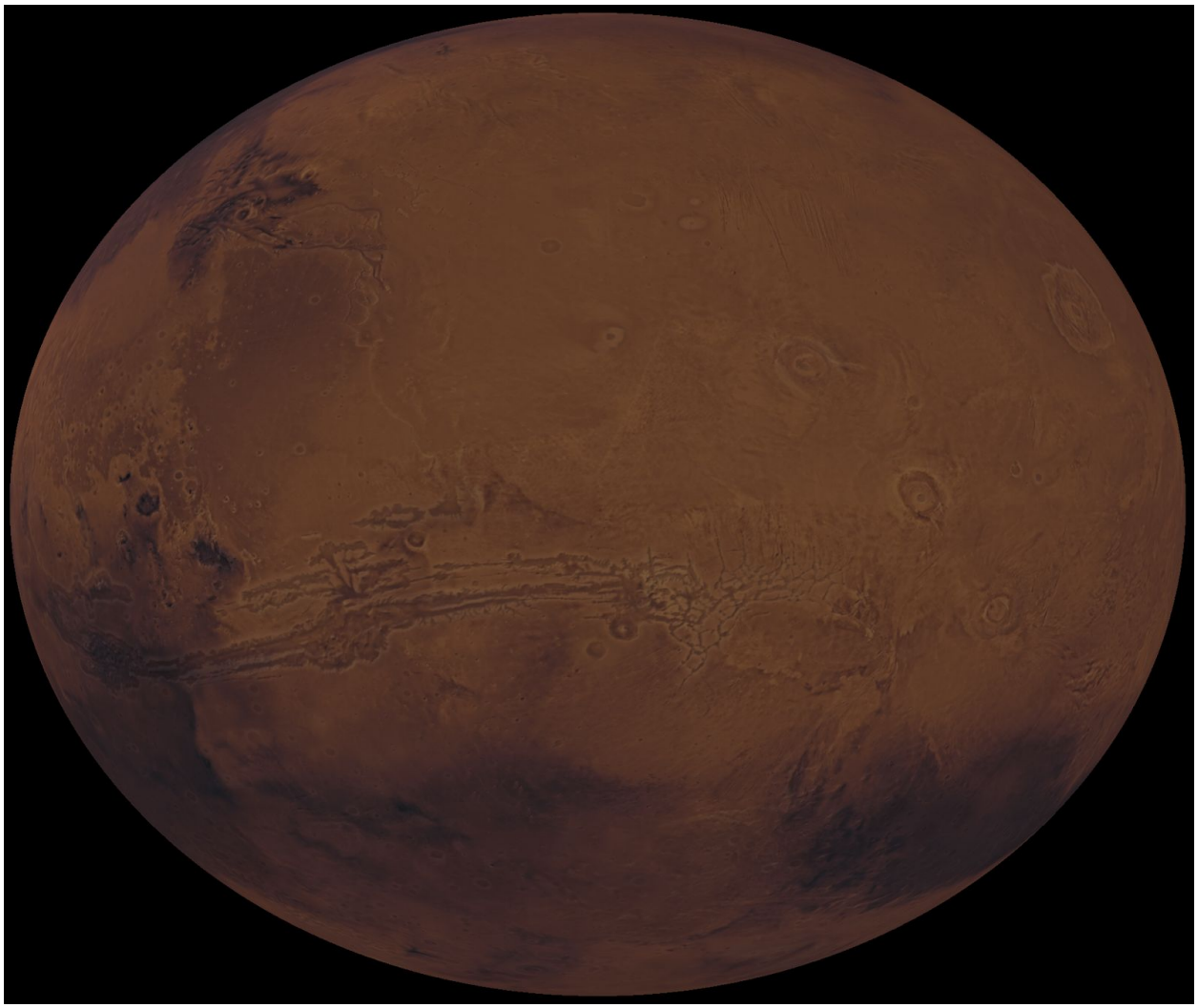
For the texture mapping I have imported a 12k texture map of Mars and used glGenTextures to upload texture. For texture minifying function GL_TEXTURE_MIN_FILTER I have used GL_NEAREST_MIPMAP_LINEAR function that picks the two mipmaps that matches the size of the pixel and uses the GL_NEAREST criterion to produce a texture value from each mipmap. For texture magnification function GL_TEXTURE_MAG_FILTER I have used GL_LINEAR function that returns the weighted average of the texture elements that are closest to the specified coordinates.

In the program I have used this texture as the surface color with texture function and get the rgb values returned from that function. Then I have binded the texture and used it for generating the Mars surface.

The screenshot of the Mars generated with this texture mapping:

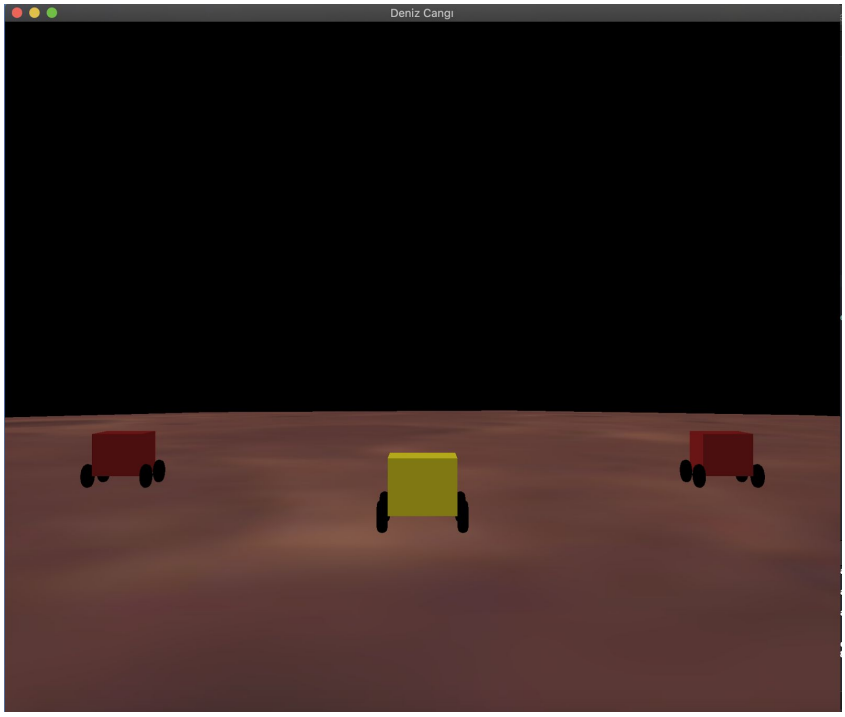


For camera I have used the Camera class implemented in learnopengl.com under the camera section. Since we want to see the rover that we will generate in the next step during the game, I set the position of it accordingly to the position glm::vec3(0.001f, 1.f+3*0.001f, 15*0.001f). With this way the camera is below the rover and it's far behind the rover. To get the view value I have used GetViewMatrix member function of the camera class which calculates the glm::lookAt of the camera with Position of the camera, Front value and Up value of the camera with glm::lookAt(Position, Position + Front, Up). Front and Up values of the camera changes as we change the camera movement. For projection I have used glm::perspective with values glm::radians(camera.Zoom) for fov, aspect as 1.f, near as 0.000001f, far as 100000.f. For fov I have used camera.Zoom since I will make the camera zoom in with the mouse scroll. Then I have get the projection * view and send this to the program to use in vertex shader for gl_Position.

For camera control I have implemented many different ways. First of all, the camera is moving with the rover when we press the UP, DOWN, RIGHT, LEFT buttons on our keyboard. When we press these buttons, the rover is moving and the camera is following the rover in these movements. Second of all, when we press the button C, we can independently move the camera without the rover. After we press button C we can move the camera with mouse cursor position all around the scene. During these movements the position of the rover does not change and the rover stays at the same position as just before we pressed the button C. To do that I have saved the position, front and up values of the camera to get back to the original position when we want to. Also during the independent camera control state, the camera can be moved with the UP, DOWN, RIGHT, LEFT buttons of the keyboard but as I have said the position of the rover remains the same and does not change. Then when we press the button V the camera goes back to the original position just before we press the button C and continues moving with the rover as it was before. I have implemented these functionalities with the member function of the Camera class which is ProcessKeyboard that gets the direction that we want to move the camera to and deltaTime that is the difference between the current time and the last time recorded. This function adjusts the camera position accordingly. During the game all the time you can zoom in, I have called the function glfwSetCursorCallback with a function that I have created as ScrollCallBack that calls one of the member function of the Camera class which is called ProcessMouseScroll which processes the input received from mouse scroll wheel on gets the vertical mouse scroll wheel input. It calculates the zoom value of the camera adjustively.

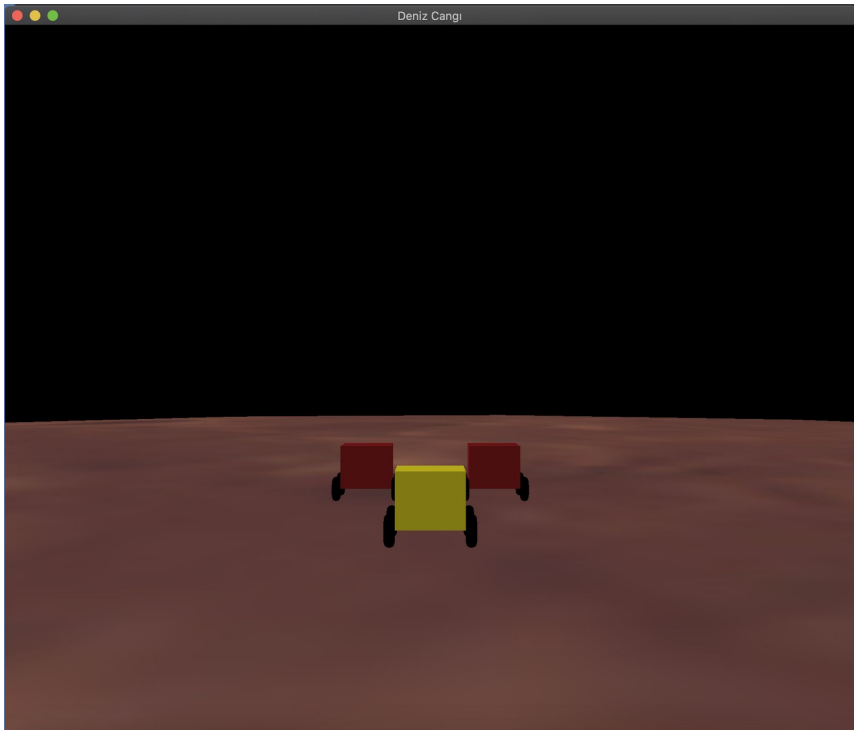
The screenshots from the scene:

When camera was moving with the rover: (before pressing the button C)



When we press the button C and move independently with camera: (With mouse cursor and the Down button to see the scene clearly.)

When we press the button V again and let the camera turn into the original position and move with rover again:



When we zoom in:

To create the rovers I have used a cube and four torus as the tires of the rover. The rover is on the top of Mars (due to the top of the Mars is white, I have rotated Mars 90 degrees so that I can play my game on a real Mars surface). The position of the player at the beginning is at glm::vec3(0.001f, 1,0001f, 0), The rover does not rotate and the scale of the rover is 0.001f. Accordingly I have calculated the transformation of the rover. When we move the rover during play mode with UP, DOWN, UP, RIGHT buttons of the keyboard, the position of the rover changes with the position of the camera accordingly as the camera follows the rover. There are 4 tires of the rover which have a parent-child relationship with the body of the rover. These tires are rotating in y axis when we move the rover forward and backward. The rover has a yellow color and when there is a collision with the enemy rovers it turns into black.

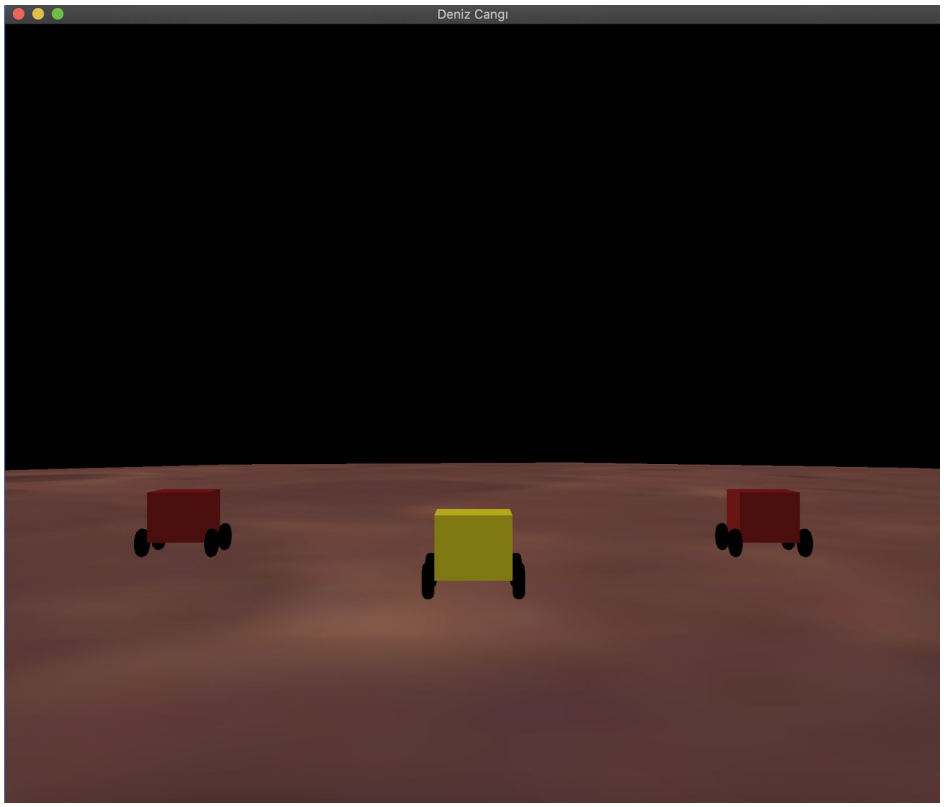
Screenshot of the rover:

The enemy rovers are implemented as cube and 4 tires too, they have red color. At the beginning they are located at the front of the rover and one has position player_position + glm::vec3(10,0,-20) * scale where player_position is the initial position of the main rover and scale is the scale of all of the rovers in the game. The other one has position player_position + glm::vec3(-10,0,-20) * scale. When the game starts they change position according to the player rover. I have calculated it with glm::mix that I use only the x and z axis of the player position and x and z axis of the enemy position since the rovers moving in the Mars in only x and z axis. The y position of the enemy does not change during the game. For linear interpolation I have used the value 0.999. Then I found the translate matrix with glm::translate, inside the translate function there is the position of the enemy and calculated the transformation of the enemy. The scale of the enemy is the same as the main rover and they are not rotating. The tires are calculated and placed exactly like the main rover.

For the collision detection I created a function called the CheckCollision and returns true if there are collisions in both x and z axis. This function takes 2 parameters, first it takes the player position and the enemy position. To check the collision in x axis I have created a boolean where it checks the player position's x axis + player_scale is greater than or equal to enemy position's x axis and enemy position's x axis + player_scale is greater than or equal to player position's x axis. If both of them are true then there is a collision in the x axis. Then check the same thing for the z axis. If there are collisions in both x and z axis then there is collision. This is checked at every iteration of the while loop and checked for both of the enemies. If there is a collision with one of the enemy rovers and main rover and the game ends. The color of the main rover turns into black and from this point of view we can not move the main rover. The colors of the enemy rovers turn into green even if any of them are able to collide with the main rover. After the collision both of the enemy rovers stop moving too. But our camera turns into independent camera mode and we can move the camera as we want with the keyboard and mouse as we did when we hit the button C during the game.

The screenshot of the enemy rovers and how the collision occurs:
The rovers moving toward the main rover when we don't move the main rover:

When we move the main rover they come after the main rover



Both of the rovers collide with the main rover and the game ends:

One of them collide with the main rover and the game ends:

After the game ends we can move the camera independently but the 3 of the rovers do not move: