FMX Animation

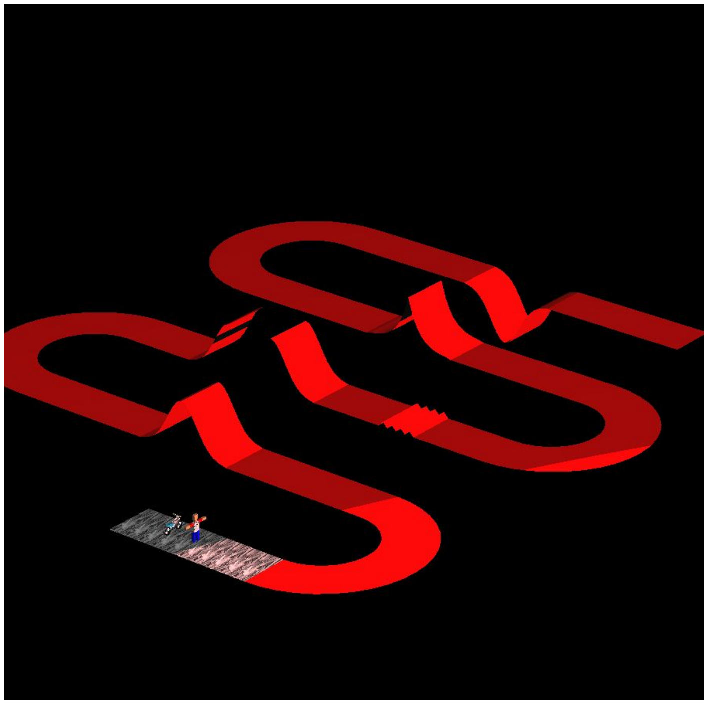
CS475 Assignment 3 Report

Akkapaka Saikiran (180050005) **Parth Vipul Sangani (18d100014)**

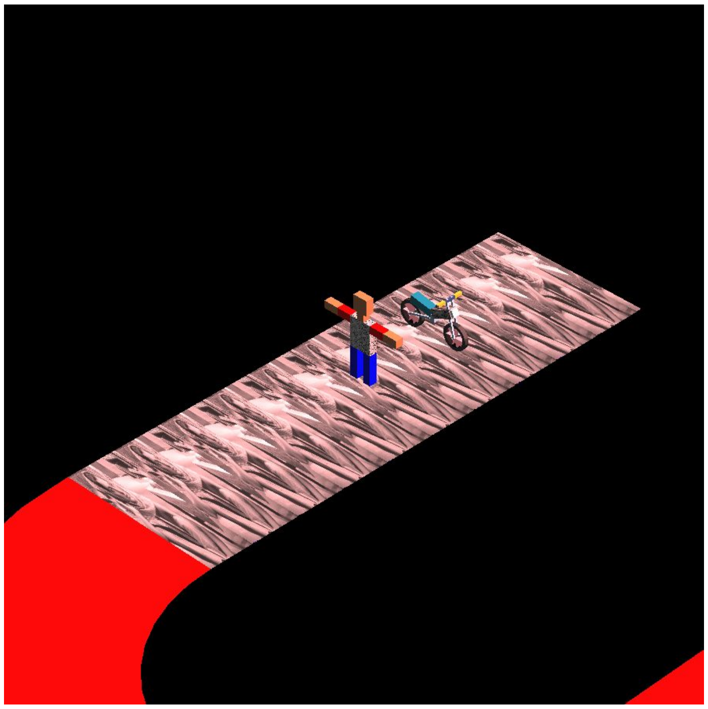
Cameras:

Three cameras have been implemented. The key ‘Z’ is used to toggle between them.

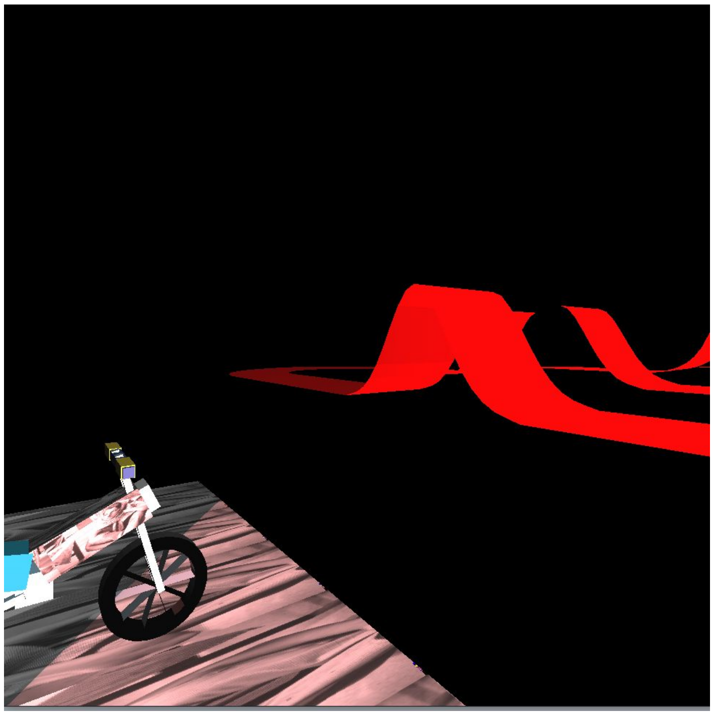
1. Camera 1 (Primary Perspective Camera) :



2. Camera 2 (Third Person Camera) :



3. Camera 3 (GoPro Camera) :



We have created a function get_transformation() which gives the product of the local transformations of the node on which this function is called. This is particularly useful in Camera Setup and Lighting.
The following example shows the usage of the function and the setting up of a camera.

```
mult = nodes["root"]->get_transformation();
mult *= nodes["hip"]->get_transformation();
lookat_pt = glm::vec3(mult * glm::vec4(0.0, 0.0, 0.0, 1.0));
glm::vec3 diff(10.0, 10.0, 10.0); // Hard coded for now
posn = lookat_pt + (diff / zoom);
```

`mult` stores the modelling transformation of the “hip” node. To get the lookat point we multiply `mult` with origin, which is the location of the hip wrt itself. The position of the camera is at a constant vector `diff` away from the lookat (module zoom). This helps us in setting up the third person camera (camera 2).

To delve into some more implementation details about cameras, we created a `Camera` class which has four `glm::vec3`s: position, rotation, up, and lookat_pt. These uniquely define the camera - its location and orientation, which are used to define the view matrix (lookat matrix). We hardcode the projection matrix accordingly for each camera. The relevant function is `loadCameras()`.

Shading:

1. Light 1 : This light is placed at (0, 60, 40) in the WCS. It changes it's position with the camera as it is defined in the VCS. It can be switched off by key K.
2. Light 2 : This light is placed at (0, 60, -40) in the WCS. It changes it's position with the camera as it is defined in the VCS. It can be switched off by key M.

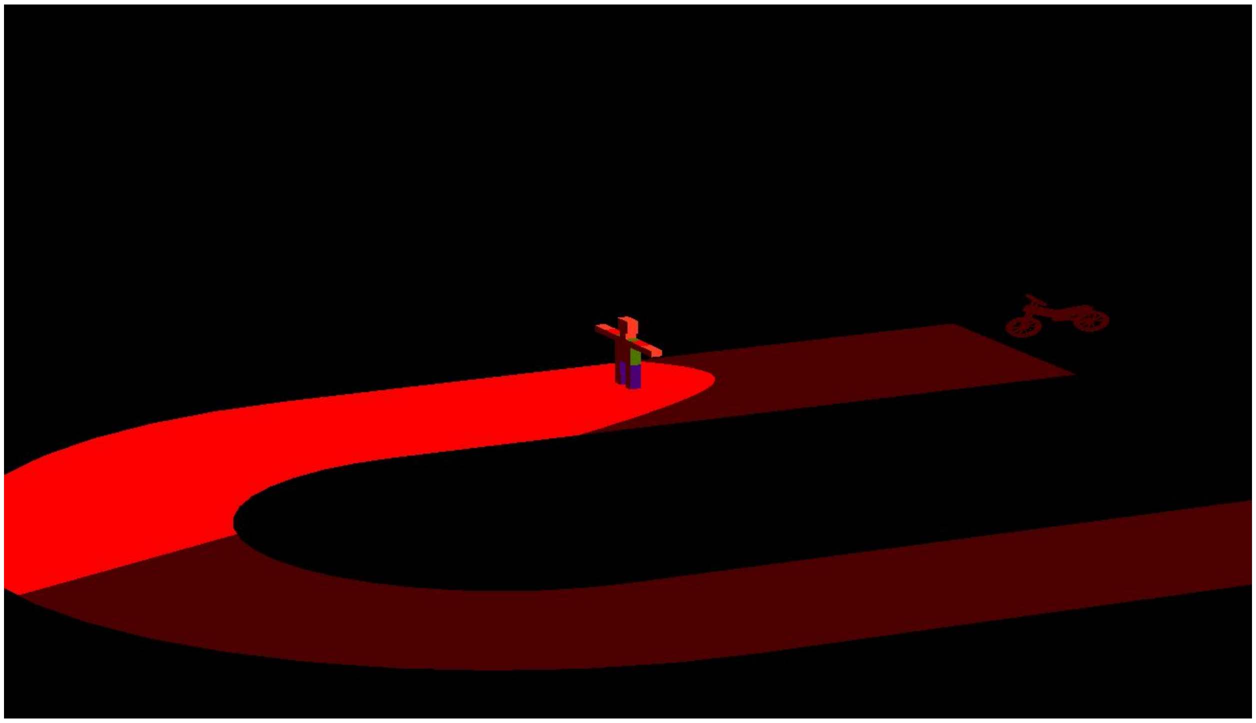
   The lights are placed on the same height but at opposite ends(which is signified by the different Z coordinate).

3. Light 3 : This light signifies the head light. The position of the headlight in the VCS is calculated in the application code and passed to the shader as a uniform vector.

```
glm::mat4 mult = nodes["root"]->get_transformation();
mult *= nodes["engine"]->get_transformation();
mult *= nodes["frontlight"]->get_transformation();
glm::vec4 posn;
posn = lookat_matrix * mult * glm::vec4(0.0, 0.0, 0.0, 1.0);
glUniform4fv(headlight, 1, glm::value_ptr(posn));
```

   It can be switched off by key O.

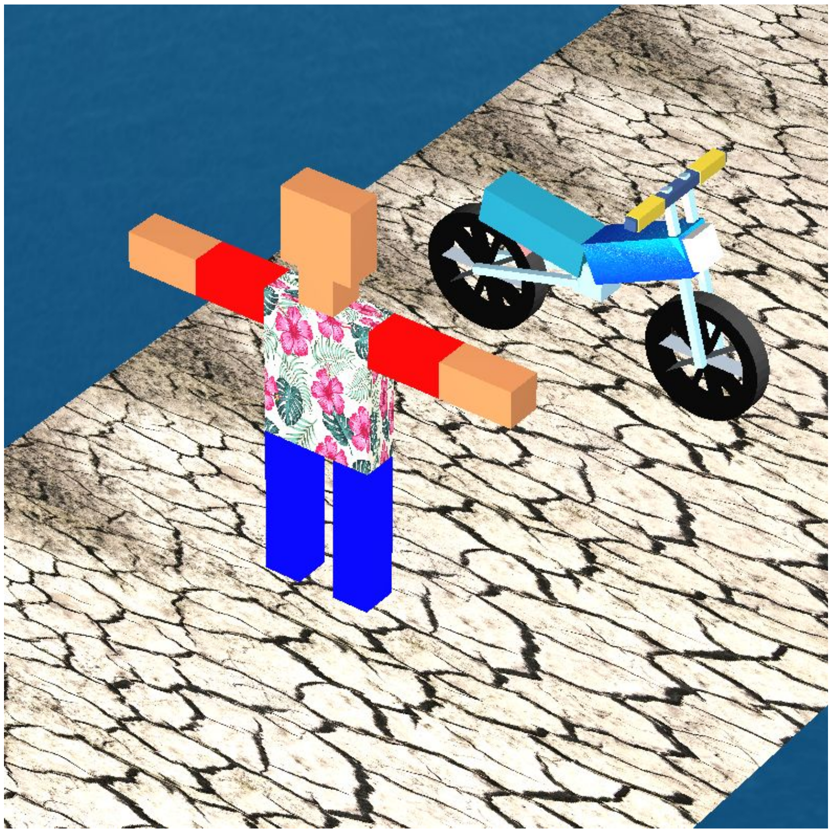
   An optimisation : We tried to design a cone around the headlight so that light only gets emitted in that cone. The code is present in the commit (normal_adjustment() function in track.cpp). However, we were facing some problems in the fragment shader due to its usage, hence we have deactivated the use of this function. The same code worked when we had not introduced texturing in the code. The screenshot attached below shows the effect we had achieved (other lights are off) -



4. Light 4 : This light signified the spotlight. The position of the spot light in the VCS is calculated in the application code and passed to the shader as a uniform vector using the get_transformation() function. It can be switched off by key Q.

Texturing:

The main idea here was to map model vertices to coordinates (texels) of some image. We accomplish this by maintaining a VBO storing texture coordinates (texcoords) for every model. These coords are passed to the fragment shader which draws colors from the concerned image using a `sampler2D` to which we apply the phong shading as described above. Thus lighting and texturing work well together. The mapping is as of now done for rectangles, and we performed texture mapping on the rider's t-shirt (torso), the bike's upper body (body1), and the straight sections of the track.
We used stb to load the images.



Environment mapping was done by creating a skybox, for which we created a separate pair of shaders. We plastered six images on the six sides of a small cube and then performed a neat trick in the fragment shader, setting the NDCS pseudo depth of the cube to 1.0, such that it is visible only when nothing is obstructing it. Thus, even though it is a small cube, it appears as though it is very far from the camera.
We borrowed texture images from babylonjs and Google Images.

Keyframing:

Key S : It saves the current state of the model on screen. It captures the camera number, light status, camera parameters and the model parameters of all the nodes in the model.

Key L : It reads the “keyframes.txt” file and reports the number of keyframes read.

Key P : It reads line 1 of the keyframe file. This is followed by a while loop which runs as long as keyframes are available. We store 2 consecutive keyframes in temporary variables. We interpolate 15 frames between consecutive keyframes (FPS=15). We have implemented wheel rotation via code instead of manually rotating the wheels in the keyframes.

Key R : It saves the interpolated frames which are generated during playback as per the format mentioned on Moodle (frame-xxxxx.tga). We have used TGA format. We have borrowed files from Assignment 1 which helps us in saving the frames. We are using the glReadPixels() function for saving the frames. Since the origin of the framebuffer is bottom left, the saved frames appear inverted. We have fixed this by rotating the video made via the “transpose” option of FFMPEG.

Frame Rate : 15

The timestamps in the “keyframes.txt” file are numbers with a constant difference of 15. Between 2 frames, we have assumed 1 second, and hence 15 frames are interpolated.

Script:

It’s practice time for Ronan the rider! He’s all pumped up to perform some wicked skills but hasn’t eaten breakfast, and more importantly, he hasn’t taken his parents’ blessings. Yet off he goes to ride. The terrain is unfamiliar to him but he is in a mood to try out new stuff. Ready, set, go! Ronan launches off at full speed.
The first obstacle is a regular old Gaussian. He’s seen it many times by now and surmounts it with ease and elegance. After a bend in the road, he comes face to face with a tougher opponent - the Gapped Gaussian. Well, Ronan’s no rookie and he manages to clear the gaping breach.
What next? A sharp speed breaker. Ronan slows down and crosses over, turning to meet the formidable Laplacian. It is here that Ronan begins to get haughty. He skillfully manoeuvres his way about the obstacle and flying off the bike with just his hands to control it. For a split second, even that control is lost. But luck is on his side today. Not for long, though.
A chilly wind blows across the track.
The final obstacle is an inverted Gaussian. It has thwarted him many times in the past. He smirks to himself. Not today. But pride goeth before a fall and Ronan Rider has a great fall. The descent is always tricky, and even though he slows down, the surface is too dusty for him to get a grip on it. Thus, he falls to his doom.
Unbeknown to him, the gods of death (Shinigami) are watching from above. They decide to toy with Ronan and give him a second chance. Or so it seems. For they know full well that time is but an illusion and events which have occurred cannot but be rolled back. The cycle of time is unrelenting with Ronan and he meets the dust again in the exact same manner.

R.I.P Ronan.
Link to the video:
 FMX Animation

https://youtu.be/RhG3SWKn6W8 (better quality)