Assignment-3+4

Lakshmi Manonmaie Pasumarthy

2^{nd} May 2019

Textures and Animation

This assignment is build upon Assignment 2, which will have 4 objects and a light which are rendered(3D-rendering).So here I am presenting the additional functionalities, which are not in Assignment 2.

- **Multiple Lights**: In the assignment 2 there is only a single light and here we have 4 lights just above the 4 objects, so we have to send an array of lights and their shades.
  You can toggle the state of light using 1,2,3,4 numbers for toggling 1^{st},2^{nd},3^{rd},4^{th} lights respectively, which track objects A,B,C,D respectively.All the models are arranged on the floor and lights are above every object.

- **Textures**: Any simple geometric object can be made kind of realistic looking wood or fur etc., without playing around with .ply file giving more and more details of specific structure. Since every vertex can be colored with only a single color, to give some design patterns on the simple geometric object, many vertices and then give colors, give a single color and then wrap it with some image of any texture you might need.This is done in Texture class, which is as follows

  - This class have name of image, width and height and also the type of image(RGB or RGBA)
  - **GenerateTexture** function will generate the required texture an assigns width and height by reading image.For loading image I have used a function stb_load from stb library.Then generate the texture and bind the textures to the buffer.Also to make the applying textures efficiently, I have used **Mipmap** which will store same image in many different sizes and thus apply the appropriate according to the size of the object.
  - **ApplyTexture** function is used to apply the texture that already have been generated

Since there is a texture, there has to be an additional buffer, to store the texture coordinates (x, y) where x,y are in range 0 to 1, to map every vertex with a point in the texture and then it will interpolate in between.We can compute texture coordinates according to how we would like the texture to be applied to any object.This is done in the function **ComputeTextureCoods** in Model.These can be:

- **Cylindrical coordinates**:In this we will image that the image is around the object rolled cylindrically and then take the texture coordinate which is just be the project of vertex of the object outwards on the cylinder.So the coordinates will be like:

$$texture\_x = (tan^{-1}(z/x) + PI)/(2 * PI)$$
$$texture\_y = ((y - min\_y)/(max\_y - min\_y)$$

  The coordinates are based on inverting the parametric coordinates.

- **Spherical coordinates**: In this the image is imagined to be on sphere around the model and the vertex is projected outward along normal direction to meet the sphere, whose coordinates will be it's texture coordinates.So the texture coordinates will be like:

$$texture\_x = (tan^{-1}(z/x) + PI)/(2 * PI)$$
$$texture\_y = (cos^{-1}(-1 * y))/PI$$

  The coordinates are based on inverting the parametric coordinates.

- **Normal mapping**:This is just mapping the image to the object just by normalizing the coordinates of the model.The equations will look like:

$$texture\_x = ((x - min\_x)/(max\_x - min\_x)$$
$$texture\_y = ((y - min\_y)/(max\_y - min\_y)$$

So once we choose the mapping, we can compute the texture coordinates for all the vertices of the model, and generate texture using **GenerateTexture** and bind the texture variables to the buffer, and then apply the texture using function **ApplyTexture**.
In the fragment shader while computing the color instead of just using the lighting,get the texture at that fragment using the computation of texture coordinates and multiply with the color we got using lighting model.

- **Animation**: So, since we have achieved how to draw patterns on the objects, without actually giving every vertex colors or increasing the number of vertices. Now we are trying to make the object move along the path we require.For this we require Scene Graph and also the paths along which object are to be moved.

  - **Scene Graph**:Since motion of some objects might depend on the motion of some other objects in the scene.The objects are to rendered in a particular order.This is represented in Scene graph.

This is a tree like structure where child depends on the position of parent, so parent is to rendered before all it's children.

- So for each model there is a vector of model pointers which point to it's children, also there is a root in view, from which the rendering is to be started. In the function **CreateSceneGraph** I am adding the dependencies to the models.
- Once the scene graph is created, it has to be iterated in a particular order, which makes sure that parent is visited before all it's children So here for iteration I have used **bfs**. And the rendered in that particular order.
- Coming to **animation**, we have to update the position of the model in each and every frame, in the path we require it to move.In the assignment the paths are given so here is how I have updated the position:
  - Since object A is stationary with respect to floor, position of object A is not updated.
  - Object B should rotate in circle around object A, so using current position of B and A, found the radius and then the angle it makes with x-axis and then using velocity and radius, found the angular velocity, and then incremented angle, using angular velocity and then found the new position on the circle with radius found and new angle computed.
  - Object C should always try to go to object B, so I have found the angle of the line BC making with X axis and then since the velocity should be in that direction, I have computed components of velocity along the x and z direction, and thus updated position (x,z).
  - Object D should stay on top of object C which means that x and z values are same for both the objects.Then it should be jumping on the object C, so I have put boundary values, which will make sure, if B goes above boundary, it must change it's direction.
  - Since all the lights are supposed to track it's objects, x and z values will be same as the object's x and z, but y will be such that the light is above the object.

- **Controller**:Almost all the controls remain same as in the Assignment 2, additionally on pressing **"m"** since we have applied textures, the mapping of the textures will cycle among cylindrical mapping, spherical mapping and normal mapping. Also on pressing **"t"** the texture we apply will vary among marble, checker board, world map and Lena.Also here the lights can toggle, by pressing 1,2,3,4 keys.In shader the computed light is added only if the state of that light is on. Here picking a object will spin that object around itself, until it is picked another time.This is done by continuously, increasing the angle, which is about the vertical axis.