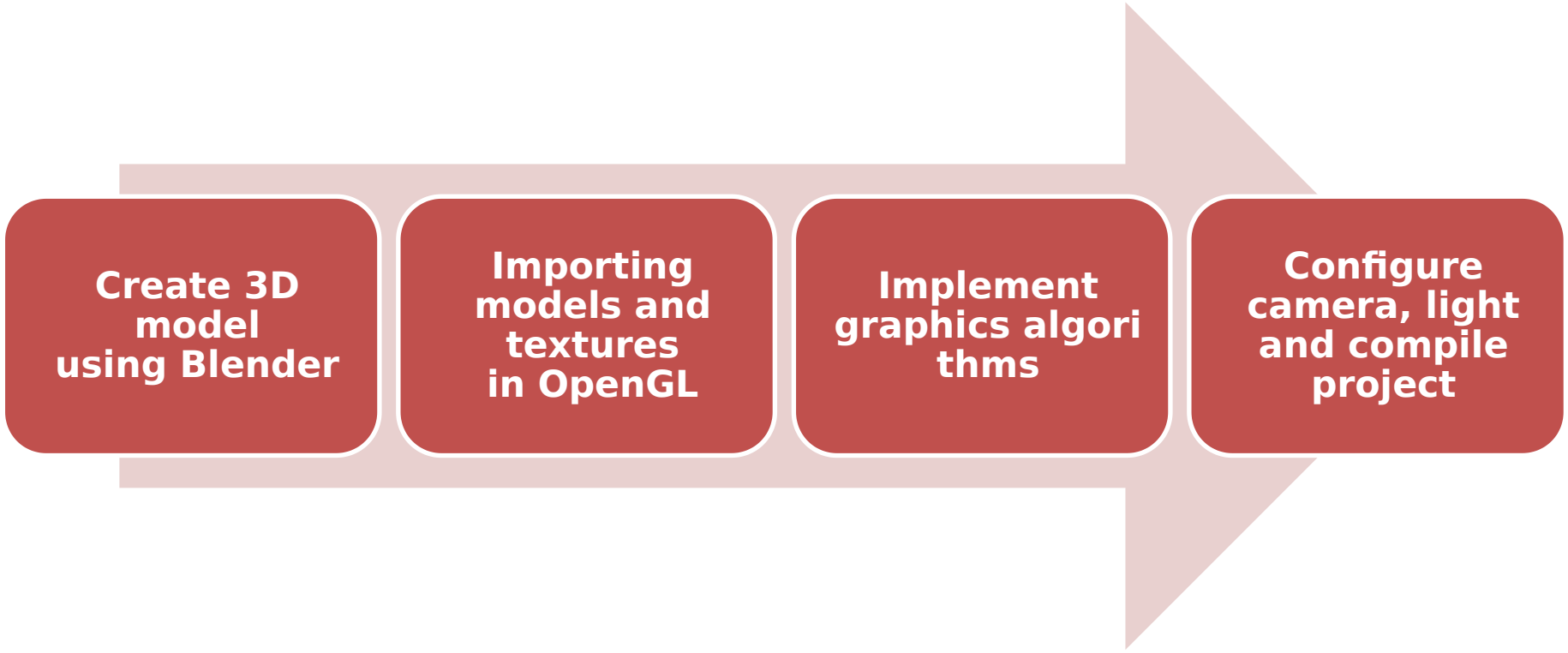
Forest road Simulation using OpenGL

Nishan Poudel(075BCT057)
Smaran Dhungana(075BCT086)
Sukriti Subedi(075BCT089)

Objectives

- To design 3D model of a forest scene.
- To learn more about graphics programming.
- To implement 3D transformations, texture filling and lighting
- To eventually make a background for a video game.

Project workflow



Technologies used

- Specification: 

- Shading: GLSL
- Editor: VS Code 
- Version control: 

- Photo Editor: 

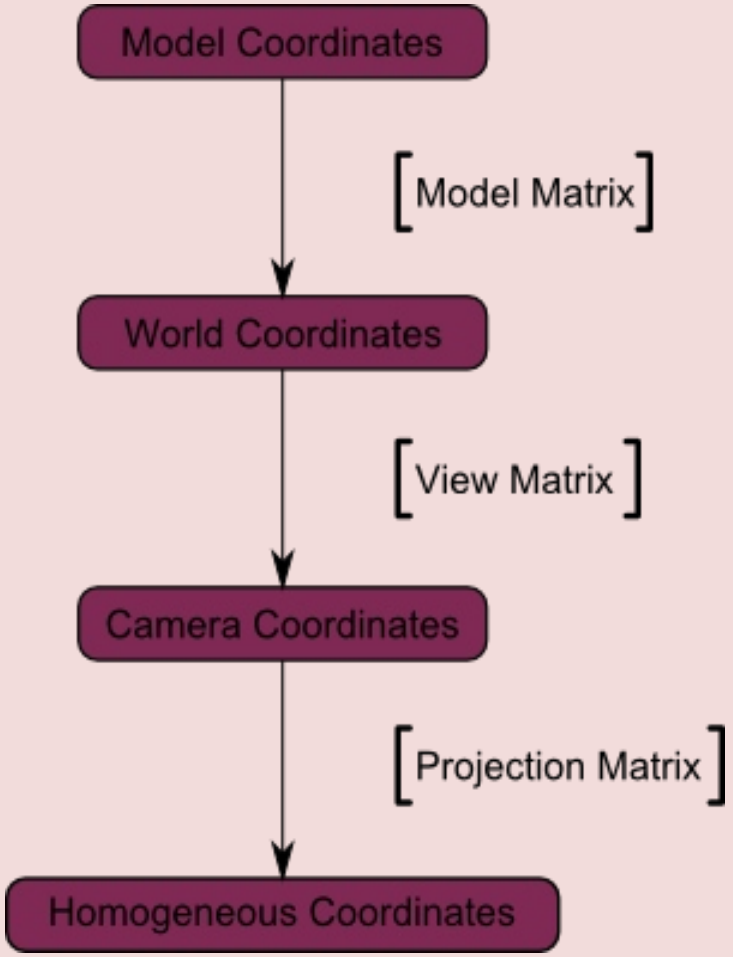
- 3D Modeling: 

- Language: 

- Build system: 

3D Viewing Pipeline

- Each element of scene is in its own model coordinates.
- Then, each element is fitted together in the scene using a Model matrix.
- After constructing the scene, all the coordinates are multiplied by a view matrix so that the viewer is at the center of the coordinate system.
- Then the coordinates in the coordinate system are multiplied by a projection matrix for realistic effects.
- Most of the transformations take place in the GPU inside the shader code.



3D Transformations

Scaling

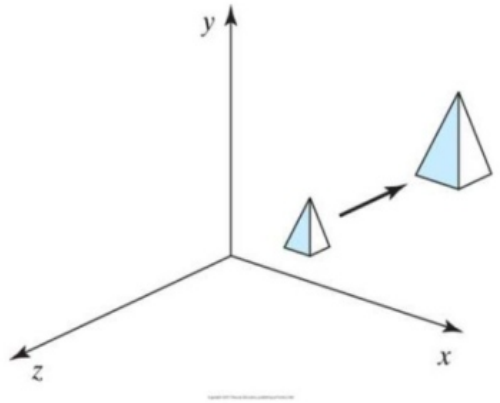
- Based on homogeneous coordinates.
- Increases or Decreases the size of a Mesh based on a reference point.

3D scaling

Figure 9-17 Doubling the size of an object with transformation 9-41 also moves the object farther from the origin.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$

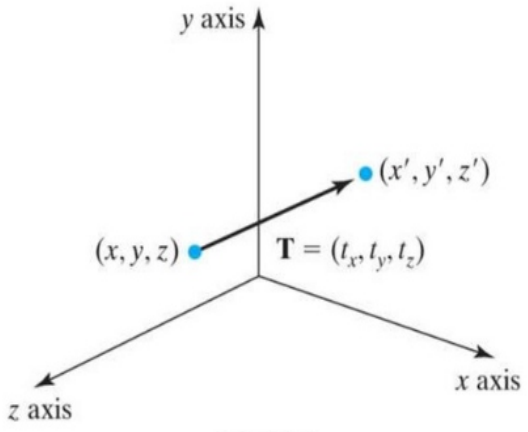


Translation

- Based on homogeneous coordinates.
- Moves object from one point in space to another.
- Translations with other transformations are called as ‘affine’

3D translation

Figure 9-1 Moving a coordinate position with translation vector $T = (t_x, t_y, t_z)$.



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

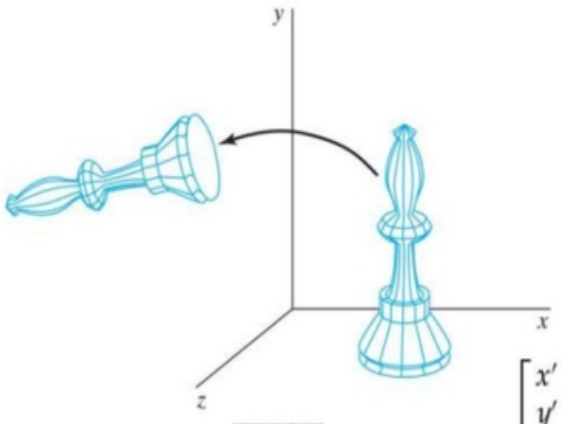
$$\mathbf{P}' = \mathbf{T} \cdot \mathbf{P}$$

Rotation

- Based on homogeneous coordinates.
- Process of rotating an object with respect to an angle in a three - dimensional plane.
- Helps seeing a scene in different angular views.

3D z-axis rotation

Figure 9-4 Rotation of an object about the z axis.



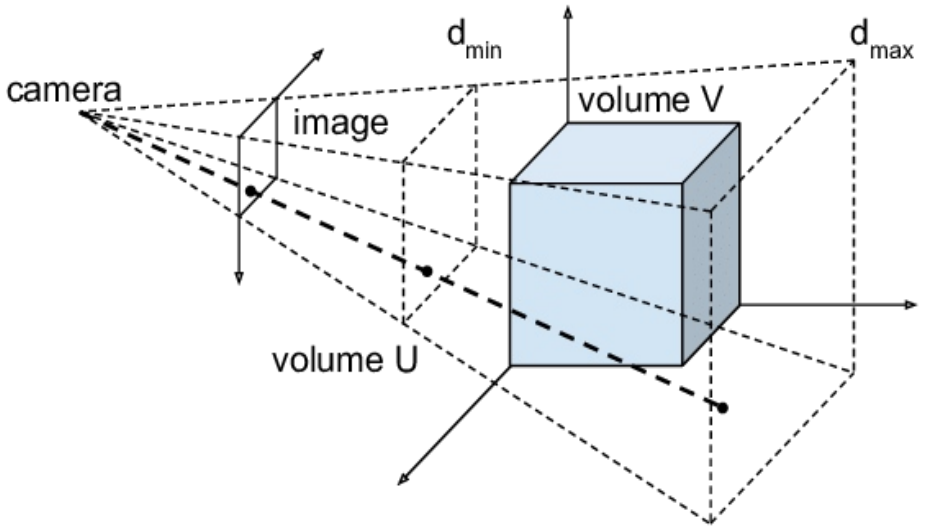
$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

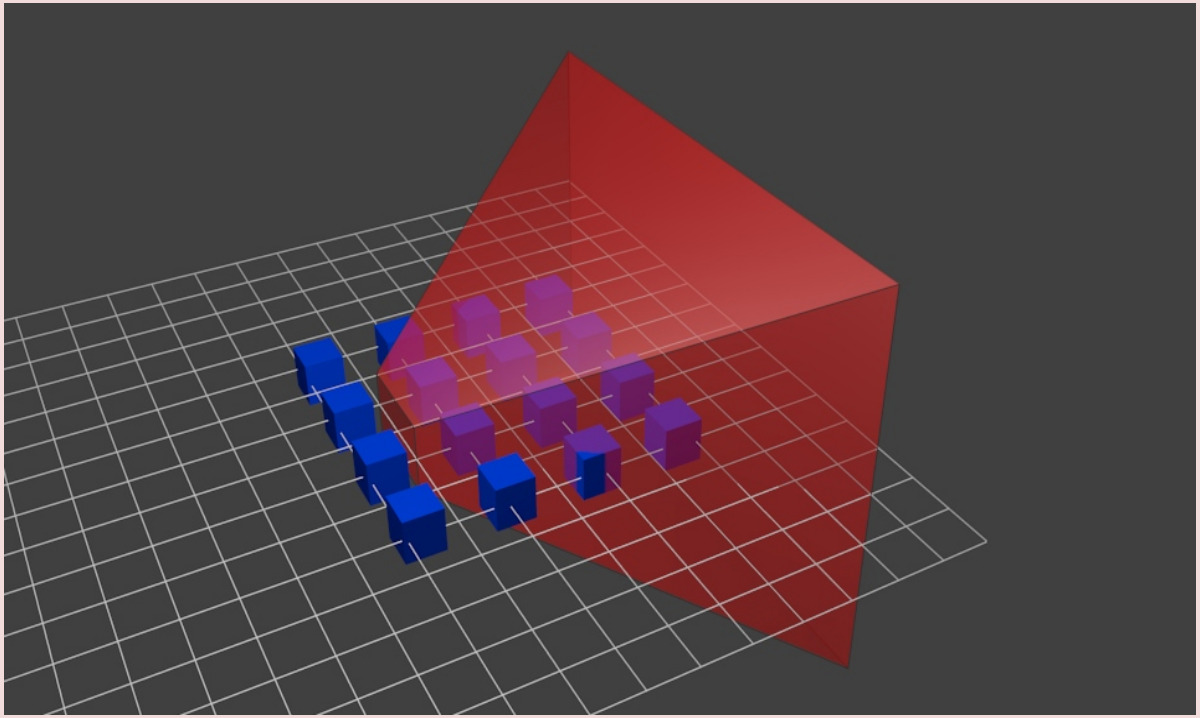
Perspective Projection

•In perspective projection, the distance from the center of projection to project plane is finite and the size of the object varies inversely with distance which looks more realistic.

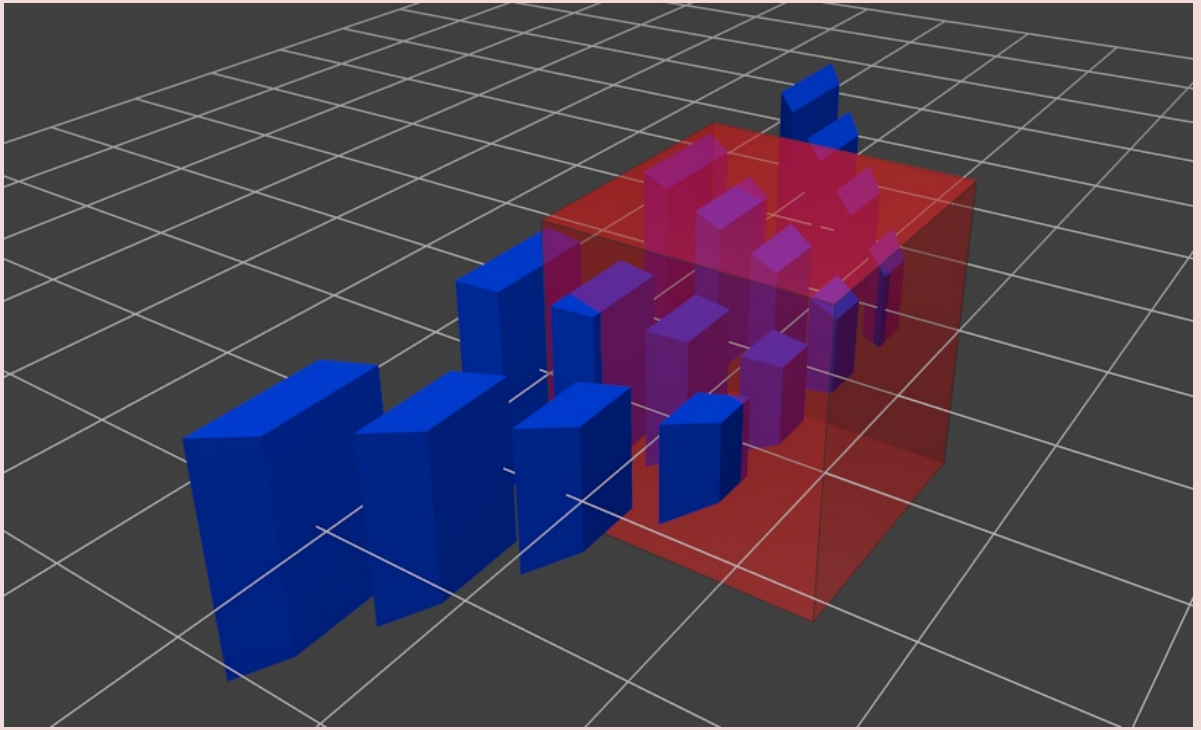
•The distance and angles are not preserved, and parallel lines do not remain parallel. Instead, they all converge at a single point called center of projection or projection reference point.



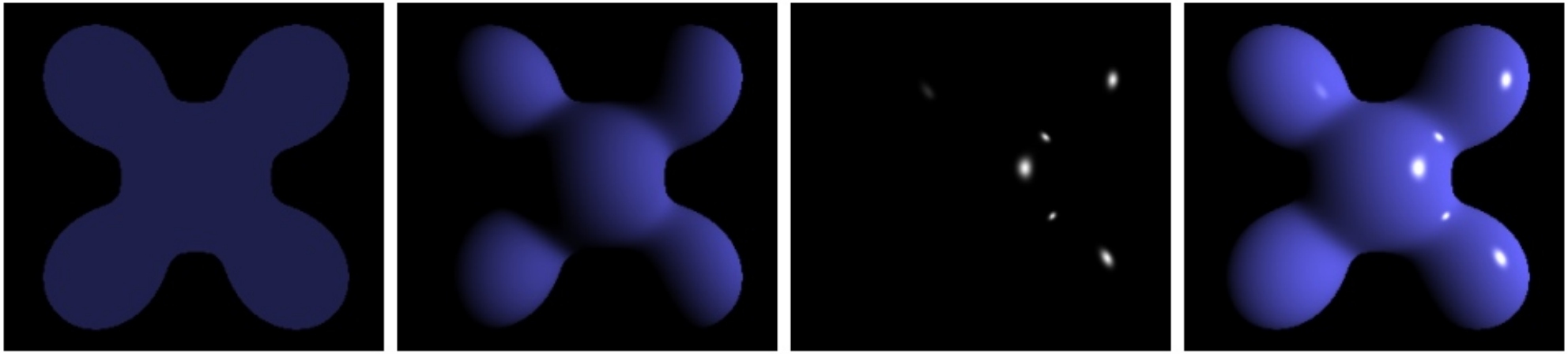
Before perspective projection



After Perspective Projection



Phong Reflection Model



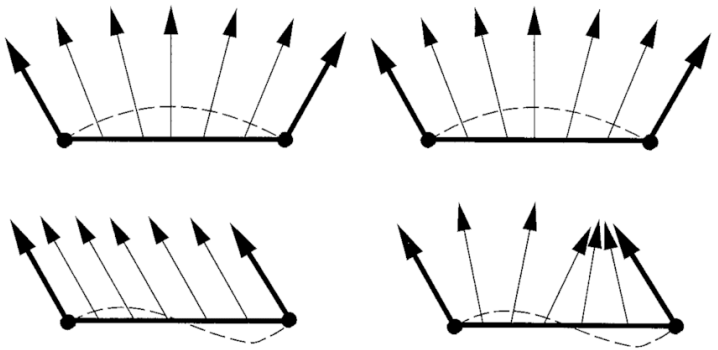
Phong Shading Model

- Phong shading is an interpolation technique for surface shading invented by the computer graphics pioneer Bui Tuong Phong.
- Phong shading improves upon Gouraud shading and provides a better approximation of the shading of a smooth surface.
- In Phong shading a normal vector is linearly interpolated across the surface of the polygon from the polygon's vertex normals. The surface normal is interpolated and normalized at each pixel and then used in a reflection model.
- Phong's methods have become the de facto baseline shading method for many rendering applications. Phong's methods have proven popular due to their generally efficient use of computation time per rendered pixel.

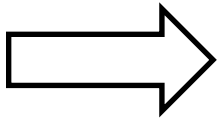
From Computer Desktop Encyclopedia
Reproduced with permission.
© 2001 Intergraph Computer Systems

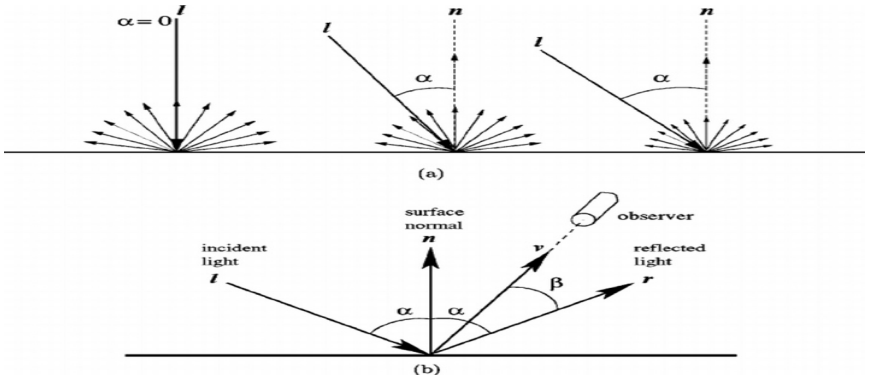
Phong shading improves upon Gouraud shading and provides a better approximation of the shading of a smooth surface.

Phong Shading Process

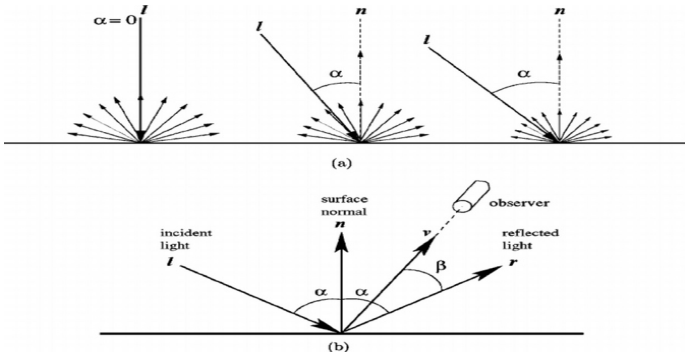


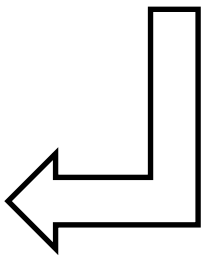
Per pixel normal calculation with interpolation.





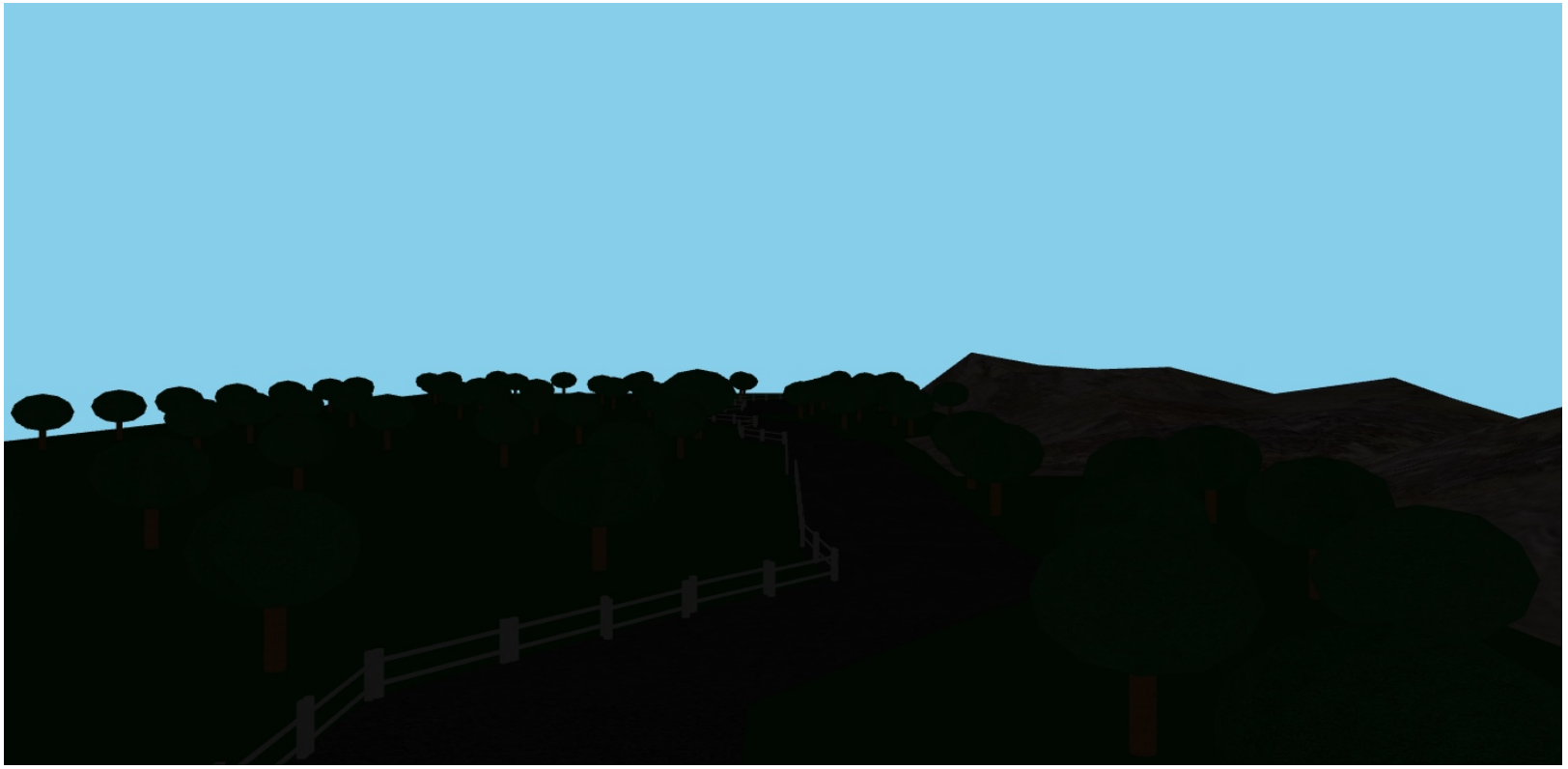
Diffuse Reflection with the help of calculated Normal



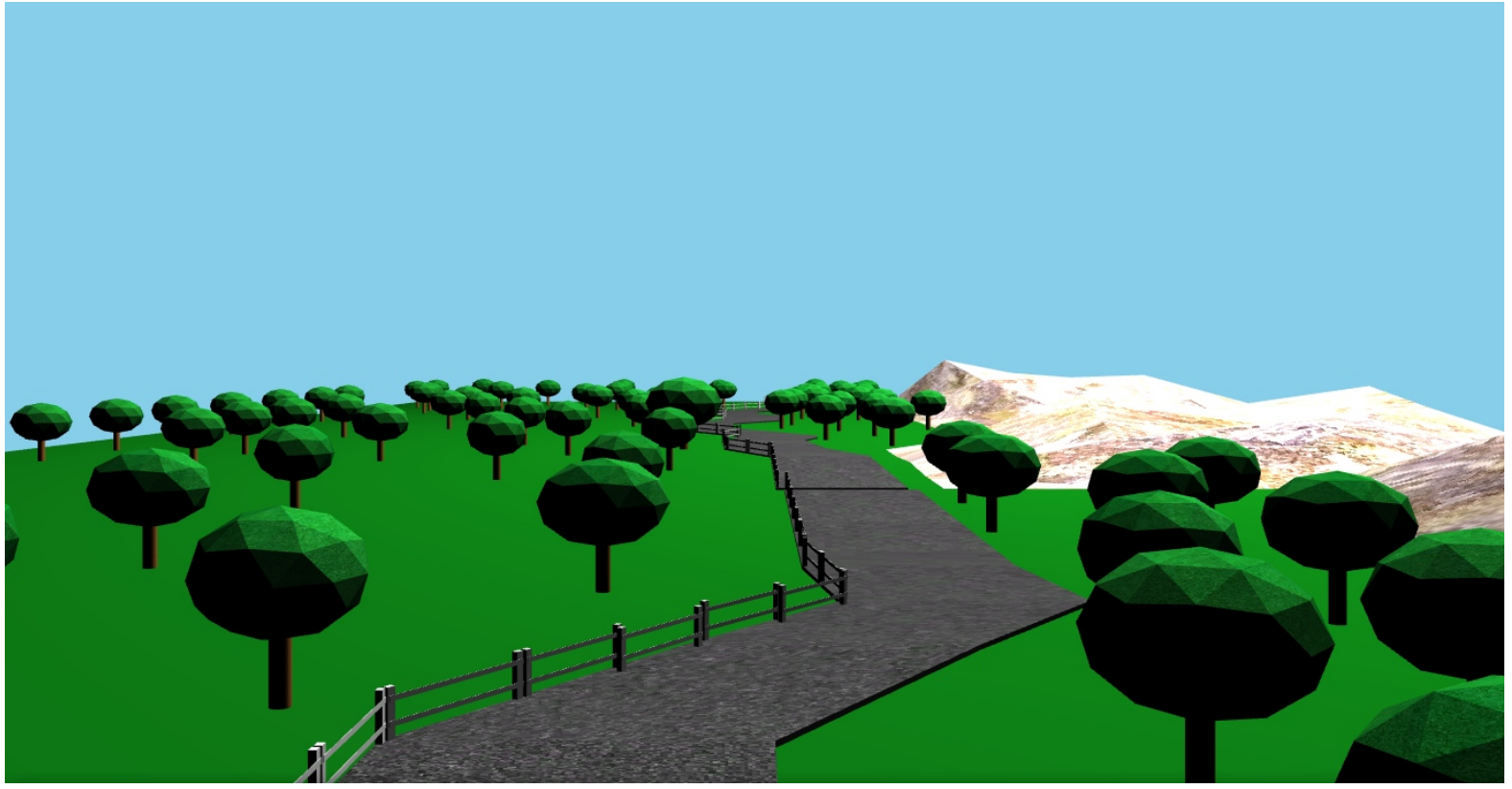


Specular Reflection with help of calculated normal, position of light and observer.

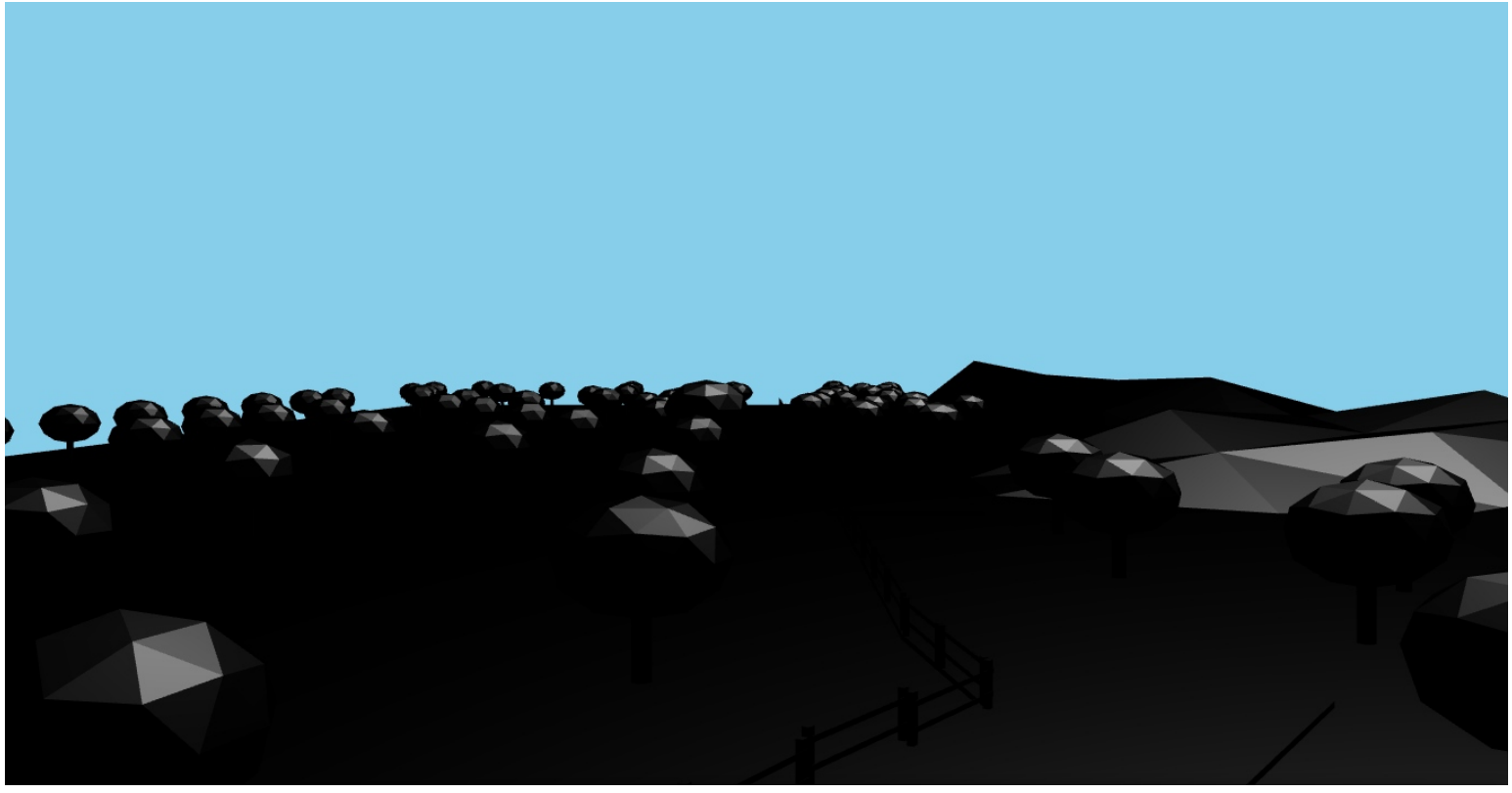
Ambient Light



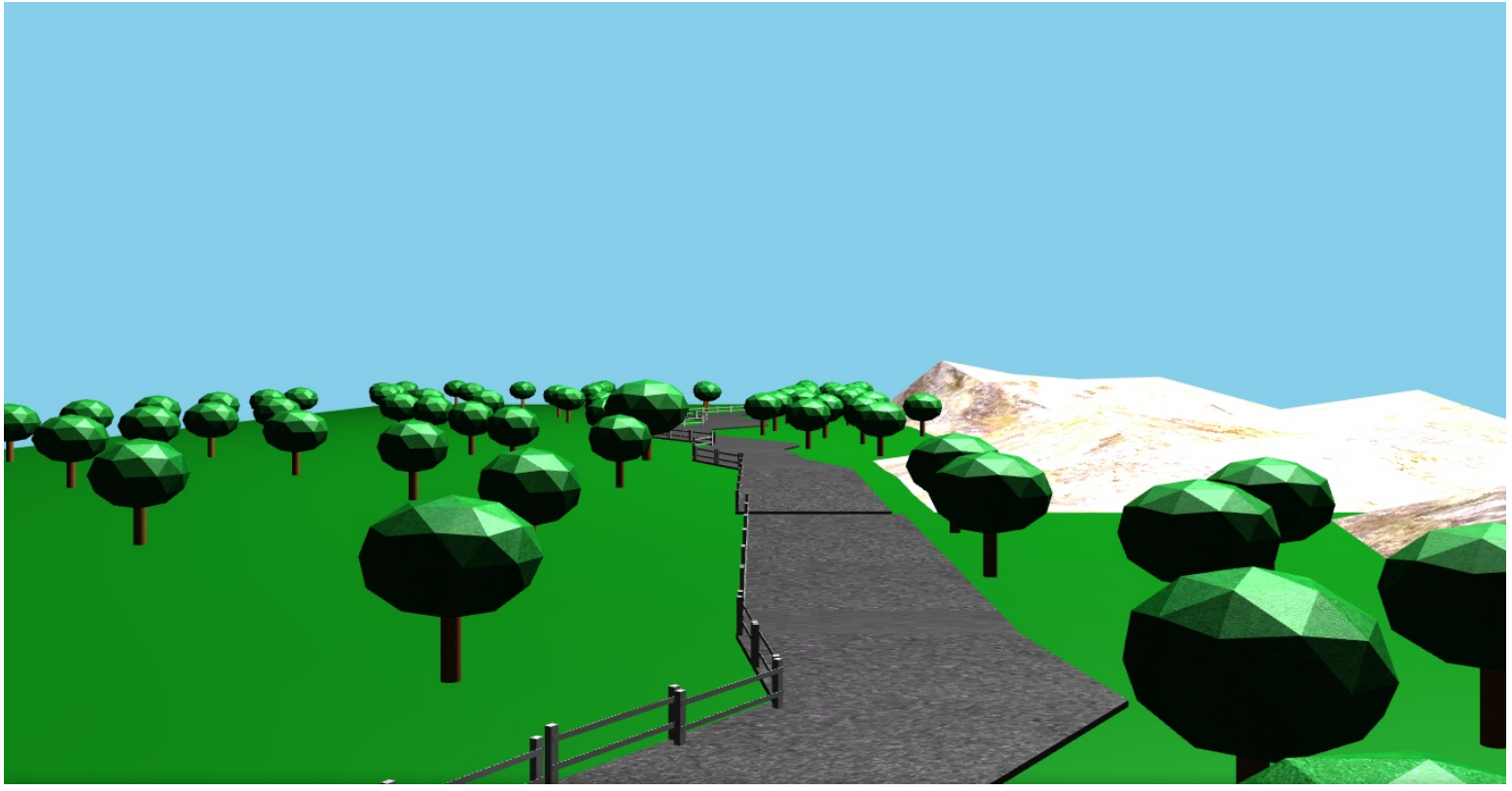
Diffuse Lighting



Specular Lighting



Combined Lighting



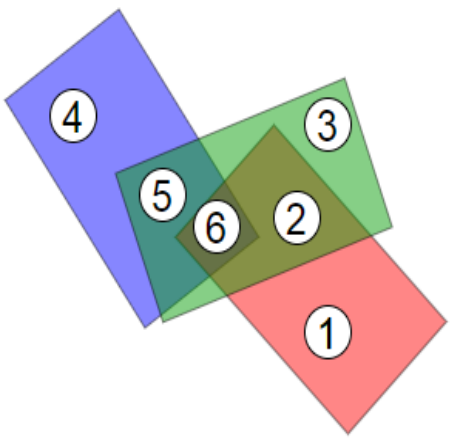
Visible Surface Detection: Z Buffer

•When viewing a picture containing non-transparent objects and surfaces, it is not possible to see those objects from view which are behind from the objects closer to eye. To get the realistic screen image, removal of these hidden surfaces is must.

•The identification and removal of these surfaces are called the Hidden-surface Problem. Z-buffer, also known as the Depth-buffer method is one of the commonly used methods for hidden surface detection.

•It is an Image space method. Image space methods are based on the pixel to be drawn on 2D. For these methods, the running time complexity is the number of pixels times number of objects. And the space complexity is two times the number of pixels because two arrays of pixels are required, one for frame buffer and the other for the depth buffer.

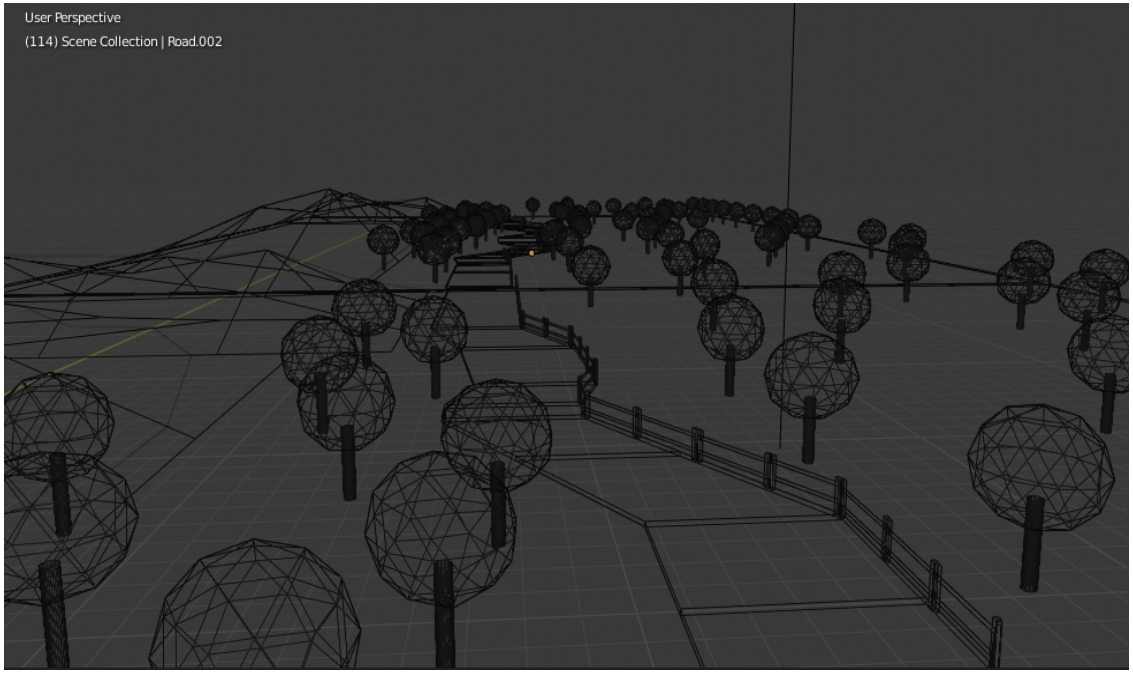
•The Z-buffer method compares surface depths at each pixel position on the projection plane.



1 Red
2 Red
3 Green
4 Blue
5 Green
6 Red

Development Phases

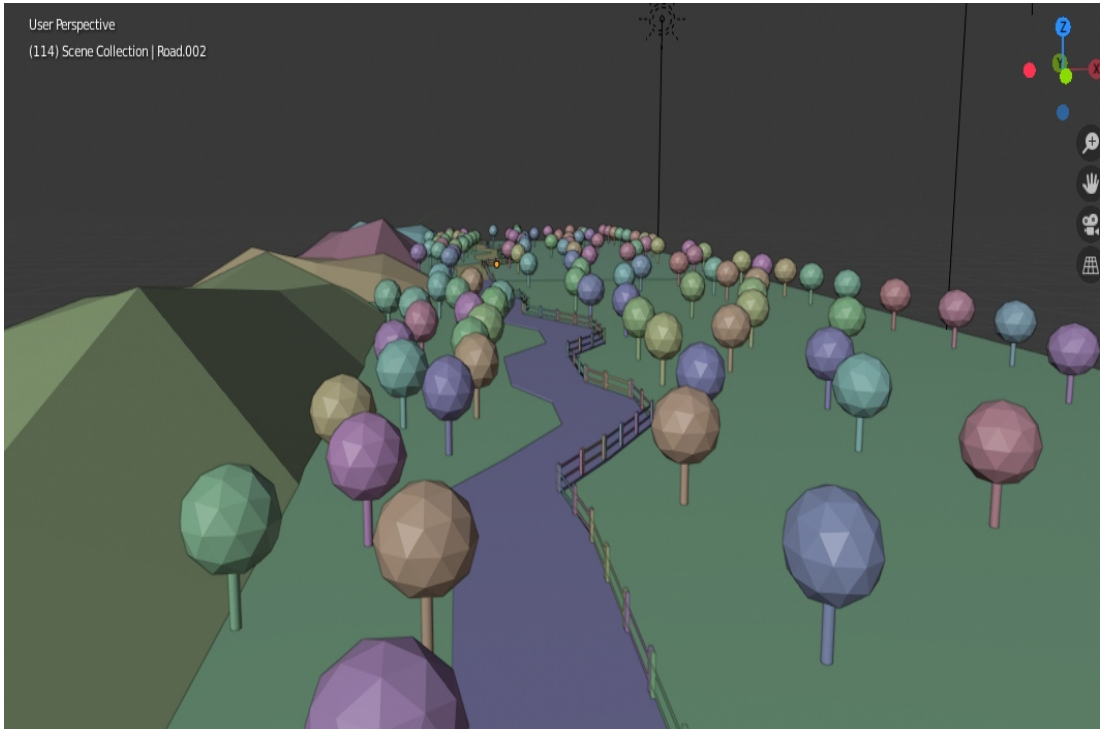
Wireframe Model



- Here, each surface is approximated by triangle as shown in the wireframe.
- Each individual plane seen in the wireframe of the model is rendered as triangle primitive in OpenGL.

Development Phases

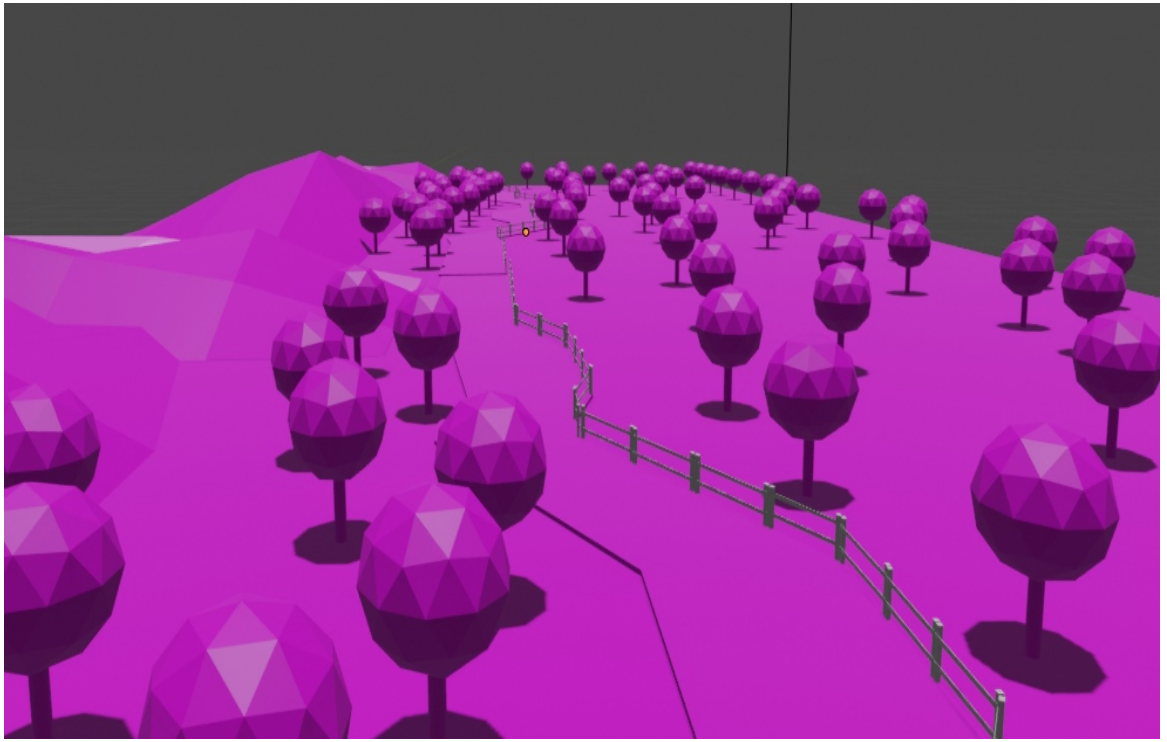
Material model



- Here, each surface is applied an appropriate texture as material property.
- Textures are initially baked from individual scene elements in Blender and later applied to each mesh.

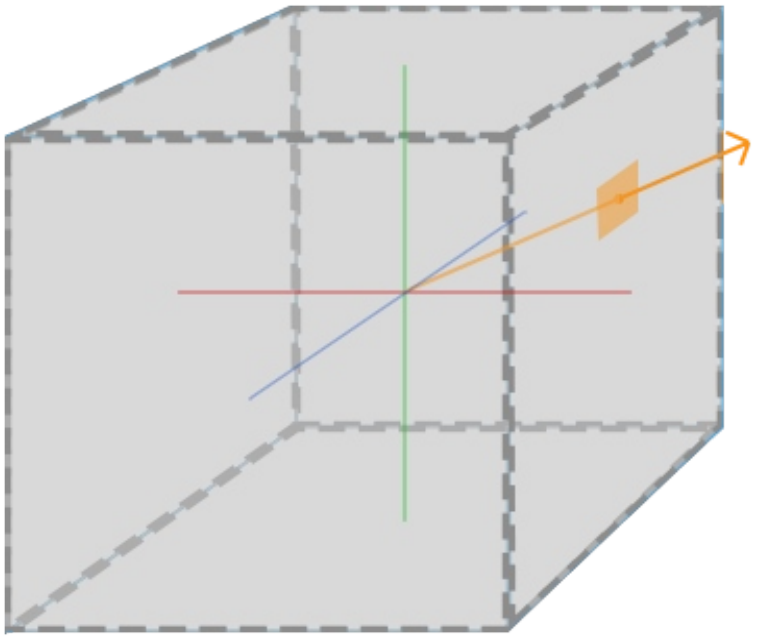
Development Phases

Viewport model

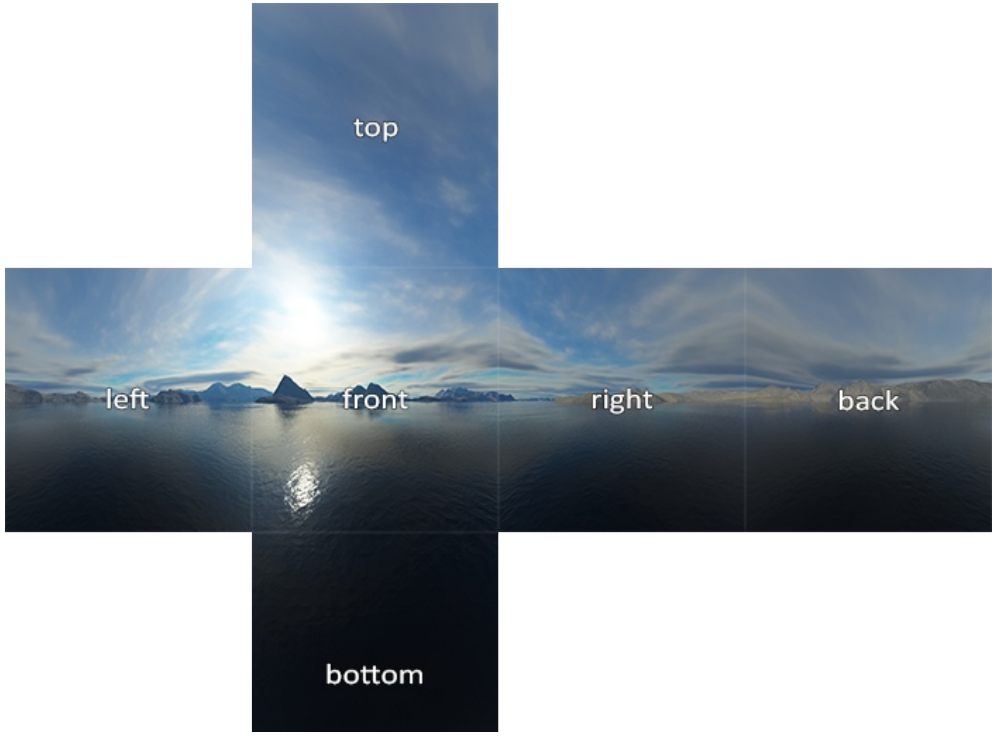


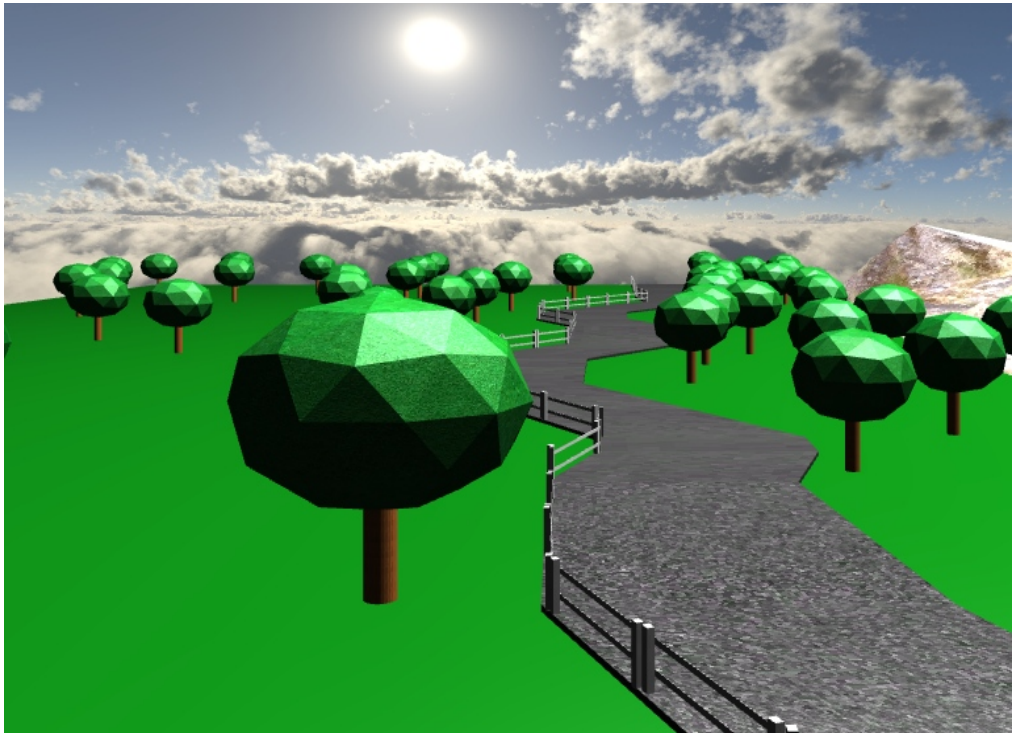
- Here, each surface has its own normal coordinate.
- The light source is simulated using a point source located above the center of the model as indicated by the shadow in the Blender viewport

Cubemap and Skybox



- A Cubemap is a texture that contains 6 individual 2D textures that each form one side of a cube: a textured cube.
- A skybox is a (large) cube that encompasses the entire scene and contains 6 images of a surrounding environment, giving the player the illusion that the environment he's in is much larger than it is.
- In order to render a skybox, we need to do the following:
  - Load the vertices of a cube into OpenGL buffers
  - Load six textures into texture objects
  - Render the cube in a scene.





Skybox texture and Scene with skybox