**UAS GRAFKOM KELOMPOK 14**

# **Petramation**

****

Anggota:

Reiner Julio -c14200136

Alvin S-c14200099

Toni A.P.-c14200121

Hans J-c14200034

Reiner Julio – C14200136 LoadObj, Laporan, Obj Kursi

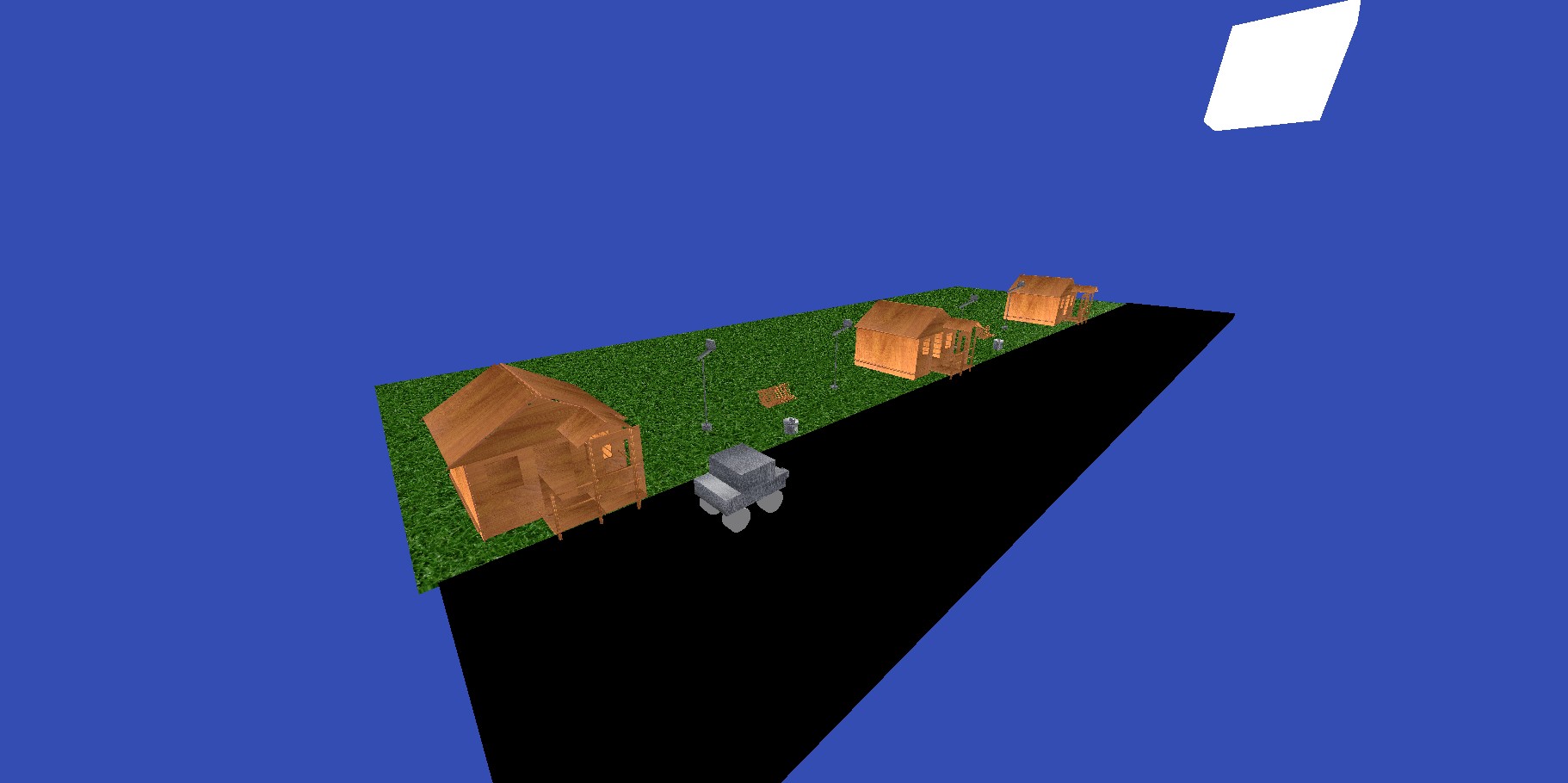
Toni Ariyanto Putra – C14200121 Camera, Video

Alvin – C14200099 Obj, LoadObj

Hans – C14200034 Shading, Lightning

Pada suatu hari di perumahan kayu yang indah dan cerah ada 1 orang yang lewat menggunakan mobil ternyata orang tersebut banyak melakukan kejahatan di daerah perumahan maka terpaksa demi menjaga keamanan hitman dikerahkan untuk membunuh penjahat menggunakan sniper.

Makna : Janganlah menjadi orang jahat karena ketika kita jahat akan ada karma yang menghatam balik pada kita

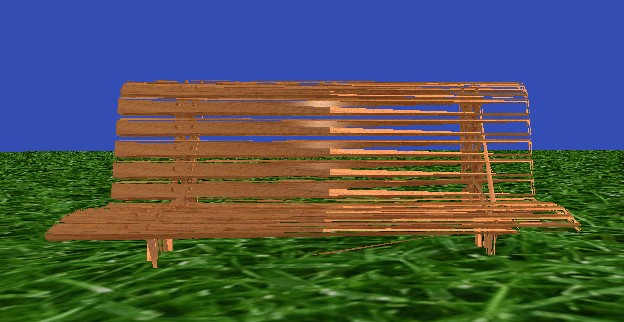












- Cara mengatur lightning sesuai story



Kami menggunakan diffuse dan juga specular serta menggunakan spotlight

Yang ditaruh di matahari agar seperti objek yang menyinari.

Berikut code yang digunakan

public void render(int mode, Camera \_camera,int dir)

{

//lighting

GL.BindVertexArray(\_VAO);

\_diffuseMap.Use(TextureUnit.Texture0);

\_specularMap.Use(TextureUnit.Texture1);

if (dir==1)

{

\_lightingShader.SetMatrix4("transform", \_transform);

\_lightingShader.SetMatrix4("view", \_camera.GetViewMatrix());

\_lightingShader.SetMatrix4("projection", \_camera.GetProjectionMatrix());

\_lightingShader.SetVector3("viewPos", \_camera.Position);

\_lightingShader.SetInt("material.diffuse", 0);

\_lightingShader.SetInt("material.specular", 1);

\_lightingShader.SetFloat("material.shininess", 64.0f);

\_lightingShader.SetVector3("light.direction", new Vector3(-0.1f, 0.7f, -2.2f));

\_lightingShader.SetVector3("light.ambient", new Vector3(0.7f));

\_lightingShader.SetVector3("light.diffuse", new Vector3(0.5f));

\_lightingShader.SetVector3("light.specular", new Vector3(0.5f, 0.5f, 0.5f));

\_lightingShader.Use();

}

else

{

\_lightingShader.SetMatrix4("transform", \_transform);

\_lightingShader.SetMatrix4("view", \_camera.GetViewMatrix());

\_lightingShader.SetMatrix4("projection", \_camera.GetProjectionMatrix());

\_lightingShader.SetVector3("viewPos", \_camera.Position);

\_lightingShader.SetInt("material.diffuse", 0);

\_lightingShader.SetInt("material.specular", 1);

\_lightingShader.SetFloat("material.shininess", 64.0f);

\_lightingShader.SetVector3("light.direction", new Vector3(0.1f, -0.7f, 2.2f));

\_lightingShader.SetVector3("light.ambient", new Vector3(0.7f));

\_lightingShader.SetVector3("light.diffuse", new Vector3(0.5f));

\_lightingShader.SetVector3("light.specular", new Vector3(0.5f, 0.5f, 0.5f));

\_lightingShader.Use();

}

switch (mode)

{

case 1:

GL.DrawArrays(PrimitiveType.Triangles, 0, vertex.Count / 8);

break;

case 2:

GL.DrawArrays(PrimitiveType.TriangleFan, 0, vertex.Count / 8);

break;

case 3:

GL.DrawArrays(PrimitiveType.LineStrip, 0, vertex.Count / 8);

break;

}

}

- Cara mengatur kamera untuk mendukung story

Kami memiliki kamera Yaw ,FoV, dna pitch serta kamera free roam,kami bisa zoom in dan juga zoom out

Berikut ialah cara untuk mengatur kamera nya

public Vector3 \_front = -Vector3.UnitZ;

private Vector3 \_up = Vector3.UnitY;

private Vector3 \_right = Vector3.UnitX;

// Rotation around the X axis (radians)

private float \_pitch;

// Rotation around the Y axis (radians)

private float \_yaw = -MathHelper.PiOver2; // Without this you would be started rotated 90 degrees right

// The field of view of the camera (radians)

private float \_fov = MathHelper.PiOver2;

public Camera(Vector3 position, float aspectRatio)

{

Position = position;

AspectRatio = aspectRatio;

}

// The position of the camera

public Vector3 Position { get; set; }

// This is simply the aspect ratio of the viewport, used for the projection matrix

public float AspectRatio { private get; set; }

public Vector3 Front => \_front;

public Vector3 Up => \_up;

public Vector3 Right => \_right;

// We convert from degrees to radians as soon as the property is set to improve performance

public float Pitch

{

get => MathHelper.RadiansToDegrees(\_pitch);

set

{

// We clamp the pitch value between -89 and 89 to prevent the camera from going upside down, and a bunch

// of weird "bugs" when you are using euler angles for rotation.

// If you want to read more about this you can try researching a topic called gimbal lock

//var angle = MathHelper.Clamp(value, -89f, 89f);

\_pitch = MathHelper.DegreesToRadians(value);

UpdateVectors();

}

}

// We convert from degrees to radians as soon as the property is set to improve performance

public float Yaw

{

get => MathHelper.RadiansToDegrees(\_yaw);

set

{

\_yaw = MathHelper.DegreesToRadians(value);

UpdateVectors();

}

}

// The field of view (FOV) is the vertical angle of the camera view, this has been discussed more in depth in a

// previous tutorial, but in this tutorial you have also learned how we can use this to simulate a zoom feature.

// We convert from degrees to radians as soon as the property is set to improve performance

public float Fov

{

get => MathHelper.RadiansToDegrees(\_fov);

set

{

var angle = MathHelper.Clamp(value, 1f, 179f);

\_fov = MathHelper.DegreesToRadians(angle);

}

}

// Get the view matrix using the amazing LookAt function described more in depth on the web tutorials

public Matrix4 GetViewMatrix()

{

return Matrix4.LookAt(Position, Position + \_front, \_up);

}

// Get the projection matrix using the same method we have used up until this point

public Matrix4 GetProjectionMatrix()

{

return Matrix4.CreatePerspectiveFieldOfView(\_fov, AspectRatio, 0.01f, 100f);

}

// This function is going to update the direction vertices using some of the math learned in the web tutorials

private void UpdateVectors()

{

// First the front matrix is calculated using some basic trigonometry

\_front.X = MathF.Cos(\_pitch) \* MathF.Cos(\_yaw);

\_front.Y = MathF.Sin(\_pitch);

\_front.Z = MathF.Cos(\_pitch) \* MathF.Sin(\_yaw);

// We need to make sure the vectors are all normalized, as otherwise we would get some funky results

\_front = Vector3.Normalize(\_front);

// Calculate both the right and the up vector using cross product

// Note that we are calculating the right from the global up, this behaviour might

// not be what you need for all cameras so keep this in mind if you do not want a FPS camera

\_right = Vector3.Normalize(Vector3.Cross(\_front, Vector3.UnitY));

\_up = Vector3.Normalize(Vector3.Cross(\_right, \_front));

}

}

}

Dan kami menggunakan command tersebut untuk menggerakkan kamera

protected override void OnUpdateFrame(FrameEventArgs args)

{

const float cameraSpeed = 0.5f;

// Escape

if (KeyboardState.IsKeyDown(Keys.Escape))

{

Close();

}

// Zoom in

if (KeyboardState.IsKeyDown(Keys.I))

{

\_camera.Fov -= 0.05f;

}

// Zoom out

if (KeyboardState.IsKeyDown(Keys.O))

{

\_camera.Fov += 0.05f;

}

// Rotasi X di pivot Camera

// Lihat ke atas (T)

if (KeyboardState.IsKeyDown(Keys.T))

{

\_camera.Pitch += 0.05f;

}

// Lihat ke bawah (G)

if (KeyboardState.IsKeyDown(Keys.G))

{

\_camera.Pitch -= 0.05f;

}

// Rotasi Y di pivot Camera

// Lihat ke kiri (F)

if (KeyboardState.IsKeyDown(Keys.F))

{

\_camera.Yaw -= 0.05f;

}

// Lihat ke kanan (H)

if (KeyboardState.IsKeyDown(Keys.H))

{

\_camera.Yaw += 0.05f;

}

// Maju (W)

if (KeyboardState.IsKeyDown(Keys.W))

{

\_camera.Position += \_camera.Front \* cameraSpeed \* (float)args.Time;

}

// Mundur (S)

if (KeyboardState.IsKeyDown(Keys.S))

{

\_camera.Position -= \_camera.Front \* cameraSpeed \* (float)args.Time;

}

// Kiri (A)

if (KeyboardState.IsKeyDown(Keys.A))

{

\_camera.Position -= \_camera.Right \* cameraSpeed \* (float)args.Time;

}

// Kanan (D)

if (KeyboardState.IsKeyDown(Keys.D))

{

\_camera.Position += \_camera.Right \* cameraSpeed \* (float)args.Time;

}

// Naik (Spasi)

if (KeyboardState.IsKeyDown(Keys.Space))

{

\_camera.Position += \_camera.Up \* cameraSpeed \* (float)args.Time;

}

// Turun (Ctrl)

if (KeyboardState.IsKeyDown(Keys.LeftControl))

{

\_camera.Position -= \_camera.Up \* cameraSpeed \* (float)args.Time;

}

if (KeyboardState.IsKeyDown(Keys.Backslash))

{

mobil.translate(0.0f, 0.0f, -cameraSpeed \* (float)args.Time);

}

if (KeyboardState.IsKeyDown(Keys.RightBracket))

{

mobil.translate(0.0f, 0.0f, cameraSpeed \* (float)args.Time);

}

if (KeyboardState.IsKeyDown(Keys.Left))

{

mobil.translate(0.0f, 0.0f, cameraSpeed \* (float)args.Time);

\_camera.Position -= \_camera.Right \* cameraSpeed \* (float)args.Time;

}

if (KeyboardState.IsKeyDown(Keys.Right))

{

mobil.translate(0.0f, 0.0f, -cameraSpeed \* (float)args.Time);

\_camera.Position += \_camera.Right \* cameraSpeed \* (float)args.Time;

}

if (KeyboardState.IsKeyDown(Keys.Up))

{

mobil.translate(-cameraSpeed \* (float)args.Time, 0.0f, 0.0f);

\_camera.Position += \_camera.Right \* cameraSpeed \* (float)args.Time;

}

if (KeyboardState.IsKeyDown(Keys.Down))

{

mobil.translate(cameraSpeed \* (float)args.Time, 0.0f, 0.0f);

\_camera.Position += \_camera.Right \* cameraSpeed \* (float)args.Time;

}

const float \_rotationSpeed = 0.02f;

// K (atas -> Rotasi sumbu x)

if (KeyboardState.IsKeyDown(Keys.K))

{

\_objectPos \*= 2;

var axis = new Vector3(1, 0, 0);

\_camera.Position -= \_objectPos;

\_camera.Pitch -= \_rotationSpeed;

\_camera.Position = Vector3.Transform(\_camera.Position,

generateArbRotationMatrix(axis, \_objectPos, \_rotationSpeed).ExtractRotation());

\_camera.Position += \_objectPos;

\_camera.\_front = -Vector3.Normalize(\_camera.Position - \_objectPos);

\_objectPos /= 2;

}

// M (bawah -> Rotasi sumbu x)

if (KeyboardState.IsKeyDown(Keys.M))

{

\_objectPos \*= 2;

var axis = new Vector3(1, 0, 0);

\_camera.Position -= \_objectPos;

\_camera.Pitch += \_rotationSpeed;

\_camera.Position = Vector3.Transform(\_camera.Position,

generateArbRotationMatrix(axis, \_objectPos, -\_rotationSpeed).ExtractRotation());

\_camera.Position += \_objectPos;

\_camera.\_front = -Vector3.Normalize(\_camera.Position - \_objectPos);

\_objectPos /= 2;

}

// N (kiri -> Rotasi sumbu y)

if (KeyboardState.IsKeyDown(Keys.N))

{

\_objectPos \*= 2;

var axis = new Vector3(0, 1, 0);

\_camera.Position -= \_objectPos;

\_camera.Yaw += \_rotationSpeed;

\_camera.Position = Vector3.Transform(\_camera.Position,

generateArbRotationMatrix(axis, \_objectPos, \_rotationSpeed).ExtractRotation());

\_camera.Position += \_objectPos;

\_camera.\_front = -Vector3.Normalize(\_camera.Position - \_objectPos);

\_objectPos /= 2;

}

// , (kanan -> Rotasi sumbu y)

if (KeyboardState.IsKeyDown(Keys.Comma))

{

\_objectPos \*= 2;

var axis = new Vector3(0, 1, 0);

\_camera.Position -= \_objectPos;

\_camera.Yaw -= \_rotationSpeed;

\_camera.Position = Vector3.Transform(\_camera.Position,

generateArbRotationMatrix(axis, \_objectPos, -\_rotationSpeed).ExtractRotation());

\_camera.Position += \_objectPos;

\_camera.\_front = -Vector3.Normalize(\_camera.Position - \_objectPos);

\_objectPos /= 2;

}

// J (putar -> Rotasi sumbu z)

if (KeyboardState.IsKeyDown(Keys.J))

{

\_objectPos \*= 2;

var axis = new Vector3(0, 0, 1);

\_camera.Position -= \_objectPos;

\_camera.Position = Vector3.Transform(\_camera.Position,

generateArbRotationMatrix(axis, \_objectPos, \_rotationSpeed).ExtractRotation());

\_camera.Position += \_objectPos;

\_camera.\_front = -Vector3.Normalize(\_camera.Position - \_objectPos);

\_objectPos /= 2;

}

// L (putar -> Rotasi sumbu z)

if (KeyboardState.IsKeyDown(Keys.L))

{

\_objectPos \*= 2;

var axis = new Vector3(0, 0, 1);

\_camera.Position -= \_objectPos;

\_camera.Position = Vector3.Transform(\_camera.Position,

generateArbRotationMatrix(axis, \_objectPos, -\_rotationSpeed).ExtractRotation());

\_camera.Position += \_objectPos;

\_camera.\_front = -Vector3.Normalize(\_camera.Position - \_objectPos);

\_objectPos /= 2;

}

if (!IsFocused)

{

return;

}

const float sensitivity = 0.02f;

if (\_firstMove)

{

\_lastMousePosition = new Vector2(MouseState.X, MouseState.Y);

\_firstMove = false;

}

else

{

// Hitung selisih mouse position

var deltaX = MouseState.X - \_lastMousePosition.X;

var deltaY = MouseState.Y - \_lastMousePosition.Y;

\_lastMousePosition = new Vector2(MouseState.X, MouseState.Y);

\_camera.Yaw += deltaX \* sensitivity;

\_camera.Pitch -= deltaY \* sensitivity;

}

- Cara mengatur collision dan meningkatkan performa

Kami tidak memiliki collision

**window.cs**

using System;

using System.Collections.Generic;

using System.Text;

using LearnOpenTK.Common;

using OpenTK.Windowing.Common;

using OpenTK.Windowing.Desktop;

using OpenTK.Graphics.OpenGL4;

using OpenTK.Mathematics;

using OpenTK.Windowing.GraphicsLibraryFramework;

namespace camera

{

class Windows : GameWindow

{

private readonly float[] \_vertices =

{

// Positions Normals Texture coords

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,

0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f,

0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,

0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,

0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,

0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,

-0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f,

-0.5f, 0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,

-0.5f, 0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f,

-0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,

-0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,

-0.5f, -0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f,

-0.5f, 0.5f, 0.5f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,

0.5f, 0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f,

0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,

0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,

0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,

0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f,

0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,

0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f,

-0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,

0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f,

0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,

-0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f

};

private readonly Vector3 \_lightPos = new Vector3(-0.1f, 0.7f,-2.2f);

private int \_vertexBufferObject;

private int \_vaoModel;

private int \_vaoLamp;

private Shader \_lampShader;

private Shader \_lightingShader;

private Camera \_camera;

private Vector3 \_objectPos;

//------------------ COBA COBA ---------------------//

Mesh pavementkiri = new Mesh();

Mesh landkiri = new Mesh();

Mesh Kandang = new Mesh();

Mesh rumahBambu = new Mesh();

Mesh orang = new Mesh();

Mesh seesaw = new Mesh();

Mesh kursi = new Mesh();

Mesh mobil = new Mesh();

Mesh lampuJalan1 = new Mesh();

Mesh lampuJalan2 = new Mesh();

Mesh lampuJalan3 = new Mesh();

Mesh lampuJalan4 = new Mesh();

Mesh tongSampah1 = new Mesh();

Mesh tongSampah2 = new Mesh();

string desertPath = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/DesertRoadPlane.obj";

string cabinetPath = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/cabinet1.obj";

string jalan = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/cobajalan.obj";

string kandang = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/kandang.obj";

string bambu = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/bamboo.obj";

string manusia = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/Male.OBJ";

string jungkat = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/seesaw.obj";

string bench = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/newBench.obj";

string ngengeng = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/mobilblender.obj";

string lampu1 = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/lampujalan1.obj";

string lampu2 = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/lampujalan2.obj";

string lampu3 = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/lampujalan3.obj";

string lampu4 = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/lampujalan4.obj";

string trashcan1 = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/trashcan1.obj";

string trashcan2 = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/trashcan2.obj";

//texture

string black = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/black.jpg";

string desertPathjpg = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/DesertRoadDiffuse.jpg";

string awanjpg = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/Clouds.jpg";

string wood = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/wood-grain-texture.jpg";

string grass = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/green-grass-texture.jpg";

string metal = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/scratched-metal-texture.jpg";

//texture

string spec1 = "C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Resources/Maps/putih.png";

private Vector2 \_lastMousePosition;

private bool \_firstMove;

public Windows(GameWindowSettings gameWindowSettings, NativeWindowSettings nativeWindowSettings) : base(gameWindowSettings, nativeWindowSettings)

{

}

private Matrix4 generateArbRotationMatrix(Vector3 axis, Vector3 center, float degree)

{

var rads = MathHelper.DegreesToRadians(degree);

var secretFormula = new float[4, 4] {

{ (float)Math.Cos(rads) + (float)Math.Pow(axis.X, 2) \* (1 - (float)Math.Cos(rads)), axis.X\* axis.Y \* (1 - (float)Math.Cos(rads)) - axis.Z \* (float)Math.Sin(rads), axis.X \* axis.Z \* (1 - (float)Math.Cos(rads)) + axis.Y \* (float)Math.Sin(rads), 0 },

{ axis.Y \* axis.X \* (1 - (float)Math.Cos(rads)) + axis.Z \* (float)Math.Sin(rads), (float)Math.Cos(rads) + (float)Math.Pow(axis.Y, 2) \* (1 - (float)Math.Cos(rads)), axis.Y \* axis.Z \* (1 - (float)Math.Cos(rads)) - axis.X \* (float)Math.Sin(rads), 0 },

{ axis.Z \* axis.X \* (1 - (float)Math.Cos(rads)) - axis.Y \* (float)Math.Sin(rads), axis.Z \* axis.Y \* (1 - (float)Math.Cos(rads)) + axis.X \* (float)Math.Sin(rads), (float)Math.Cos(rads) + (float)Math.Pow(axis.Z, 2) \* (1 - (float)Math.Cos(rads)), 0 },

{ 0, 0, 0, 1}

};

var secretFormulaMatrix = new Matrix4(

new Vector4(secretFormula[0, 0], secretFormula[0, 1], secretFormula[0, 2], secretFormula[0, 3]),

new Vector4(secretFormula[1, 0], secretFormula[1, 1], secretFormula[1, 2], secretFormula[1, 3]),

new Vector4(secretFormula[2, 0], secretFormula[2, 1], secretFormula[2, 2], secretFormula[2, 3]),

new Vector4(secretFormula[3, 0], secretFormula[3, 1], secretFormula[3, 2], secretFormula[3, 3])

);

return secretFormulaMatrix;

}

protected override void OnLoad()

{

GL.ClearColor(0.2f,0.3f,0.7f,1f);

GL.Enable(EnableCap.DepthTest);

\_vertexBufferObject = GL.GenBuffer();

GL.BindBuffer(BufferTarget.ArrayBuffer, \_vertexBufferObject);

GL.BufferData(BufferTarget.ArrayBuffer, \_vertices.Length \* sizeof(float), \_vertices, BufferUsageHint.StaticDraw);

\_lightingShader = new Shader("C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Shaders/shader.vert",

"C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Shaders/lighting.frag");

\_lampShader = new Shader("C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Shaders/shader.vert",

"C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Shaders/shader\_white.frag");

\_vaoModel = GL.GenVertexArray();

GL.BindVertexArray(\_vaoModel);

var vertexLocation = \_lightingShader.GetAttribLocation("aPosition");

GL.EnableVertexAttribArray(vertexLocation);

GL.VertexAttribPointer(vertexLocation, 3, VertexAttribPointerType.Float, false, 8 \* sizeof(float), 0);

var normalLocation = \_lightingShader.GetAttribLocation("aNormal");

GL.EnableVertexAttribArray(normalLocation);

GL.VertexAttribPointer(normalLocation, 3, VertexAttribPointerType.Float, false, 8 \* sizeof(float), 3 \* sizeof(float));

var texCoordLocation = \_lightingShader.GetAttribLocation("aTexCoords");

GL.EnableVertexAttribArray(texCoordLocation);

GL.VertexAttribPointer(texCoordLocation, 2, VertexAttribPointerType.Float, false, 8 \* sizeof(float), 6 \* sizeof(float));

\_vaoLamp = GL.GenVertexArray();

GL.BindVertexArray(\_vaoLamp);

vertexLocation = \_lampShader.GetAttribLocation("aPosition");

GL.EnableVertexAttribArray(vertexLocation);

GL.VertexAttribPointer(vertexLocation, 3, VertexAttribPointerType.Float, false, 8 \* sizeof(float), 0);

var \_cameraPosInit = new Vector3(0f, 0.5f, 0);

\_camera = new Camera(\_cameraPosInit, Size.X / (float)Size.Y);

\_camera.Yaw -= 90f;

CursorGrabbed = true;

// NGERAKIT OBJ

pavementkiri.setupObject(jalan, black, spec1, (float)Size.X, (float)Size.Y);

landkiri.setupObject(path15, grass, spec1, (float)Size.X, (float)Size.Y);

//Kandang.setupObject(kandang, wood, spec1, (float)Size.X, (float)Size.Y);

rumahBambu.setupObject(bambu, wood, spec1, (float)Size.X, (float)Size.Y);

//seesaw.setupObject(jungkat, wood, spec1 , (float)Size.X, (float)Size.Y);

kursi.setupObject(bench, wood, spec1 , (float)Size.X, (float)Size.Y);

//orang.setupObject(manusia,text1,spec1, (float)Size.X, (float)Size.Y);

//cabinet.setupObject(path21, text1, spec1, (float)Size.X, (float)Size.Y);

mobil.setupObject(ngengeng, metal, spec1 , (float)Size.X, (float)Size.Y);

lampuJalan1.setupObject(lampu1, metal, spec1, (float)Size.X, (float)Size.Y);

lampuJalan2.setupObject(lampu2, metal, spec1, (float)Size.X, (float)Size.Y);

lampuJalan3.setupObject(lampu3, metal, spec1, (float)Size.X, (float)Size.Y);

lampuJalan4.setupObject(lampu4, metal, spec1, (float)Size.X, (float)Size.Y);

tongSampah1.setupObject(trashcan1, metal, spec1, (float)Size.X, (float)Size.Y);

tongSampah2.setupObject(trashcan2, metal, spec1, (float)Size.X, (float)Size.Y);

base.OnLoad();

}

protected override void OnRenderFrame(FrameEventArgs args)

{

GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

//--------------------------------------- LOAD OBJ PATH --------------------------//

pavementkiri.render(1, \_camera, 1);

landkiri.render(1, \_camera,2);

//Kandang.render(1, \_camera, 2);

// cabinet.render(1, \_camera, 1);

rumahBambu.render(1, \_camera, 2);

//orang.render(1, \_camera, 1);

//seesaw.render(1, \_camera, 1);

kursi.render(1, \_camera, 2);

mobil.render(1, \_camera, 2);

lampuJalan1.render(1, \_camera, 2);

lampuJalan2.render(1, \_camera, 2);

lampuJalan3.render(1, \_camera, 2);

lampuJalan4.render(1, \_camera, 2);

tongSampah1.render(2, \_camera, 2);

tongSampah2.render(2, \_camera, 2);

GL.DrawArrays(PrimitiveType.Triangles, 0, 36);

GL.BindVertexArray(\_vaoModel);

Matrix4 lampMatrix = Matrix4.CreateScale(0.25f);

lampMatrix = lampMatrix \* Matrix4.CreateTranslation(\_lightPos);

\_lampShader.SetMatrix4("transform", lampMatrix);

\_lampShader.SetMatrix4("view", \_camera.GetViewMatrix());

\_lampShader.SetMatrix4("projection", \_camera.GetProjectionMatrix());

\_lampShader.Use();

GL.BindVertexArray(\_vaoLamp);

GL.DrawArrays(PrimitiveType.Triangles, 0, 36);

SwapBuffers();

base.OnRenderFrame(args);

}

protected override void OnUpdateFrame(FrameEventArgs args)

{

const float cameraSpeed = 0.5f;

// Escape

if (KeyboardState.IsKeyDown(Keys.Escape))

{

Close();

}

// Zoom in

if (KeyboardState.IsKeyDown(Keys.I))

{

\_camera.Fov -= 0.05f;

}

// Zoom out

if (KeyboardState.IsKeyDown(Keys.O))

{

\_camera.Fov += 0.05f;

}

// Rotasi X di pivot Camera

// Lihat ke atas (T)

if (KeyboardState.IsKeyDown(Keys.T))

{

\_camera.Pitch += 0.05f;

}

// Lihat ke bawah (G)

if (KeyboardState.IsKeyDown(Keys.G))

{

\_camera.Pitch -= 0.05f;

}

// Rotasi Y di pivot Camera

// Lihat ke kiri (F)

if (KeyboardState.IsKeyDown(Keys.F))

{

\_camera.Yaw -= 0.05f;

}

// Lihat ke kanan (H)

if (KeyboardState.IsKeyDown(Keys.H))

{

\_camera.Yaw += 0.05f;

}

// Maju (W)

if (KeyboardState.IsKeyDown(Keys.W))

{

\_camera.Position += \_camera.Front \* cameraSpeed \* (float)args.Time;

}

// Mundur (S)

if (KeyboardState.IsKeyDown(Keys.S))

{

\_camera.Position -= \_camera.Front \* cameraSpeed \* (float)args.Time;

}

// Kiri (A)

if (KeyboardState.IsKeyDown(Keys.A))

{

\_camera.Position -= \_camera.Right \* cameraSpeed \* (float)args.Time;

}

// Kanan (D)

if (KeyboardState.IsKeyDown(Keys.D))

{

\_camera.Position += \_camera.Right \* cameraSpeed \* (float)args.Time;

}

// Naik (Spasi)

if (KeyboardState.IsKeyDown(Keys.Space))

{

\_camera.Position += \_camera.Up \* cameraSpeed \* (float)args.Time;

}

// Turun (Ctrl)

if (KeyboardState.IsKeyDown(Keys.LeftControl))

{

\_camera.Position -= \_camera.Up \* cameraSpeed \* (float)args.Time;

}

if (KeyboardState.IsKeyDown(Keys.Backslash))

{

mobil.translate(0.0f, 0.0f, -cameraSpeed \* (float)args.Time);

}

if (KeyboardState.IsKeyDown(Keys.RightBracket))

{

mobil.translate(0.0f, 0.0f, cameraSpeed \* (float)args.Time);

}

if(KeyboardState.IsKeyDown(Keys.Left))

{

mobil.translate(0.0f, 0.0f, cameraSpeed \* (float)args.Time);

\_camera.Position -= \_camera.Right \* cameraSpeed \* (float)args.Time;

}

if(KeyboardState.IsKeyDown(Keys.Right))

{

mobil.translate(0.0f, 0.0f, -cameraSpeed \* (float)args.Time);

\_camera.Position += \_camera.Right \* cameraSpeed \* (float)args.Time;

}

if (KeyboardState.IsKeyDown(Keys.Up))

{

mobil.translate(-cameraSpeed \* (float)args.Time, 0.0f, 0.0f);

\_camera.Position += \_camera.Right \* cameraSpeed \* (float)args.Time;

}

if (KeyboardState.IsKeyDown(Keys.Down))

{

mobil.translate(cameraSpeed \* (float)args.Time,0.0f, 0.0f);

\_camera.Position += \_camera.Right \* cameraSpeed \* (float)args.Time;

}

const float \_rotationSpeed = 0.02f;

// K (atas -> Rotasi sumbu x)

if (KeyboardState.IsKeyDown(Keys.K))

{

\_objectPos \*= 2;

var axis = new Vector3(1, 0, 0);

\_camera.Position -= \_objectPos;

\_camera.Pitch -= \_rotationSpeed;

\_camera.Position = Vector3.Transform(\_camera.Position,

generateArbRotationMatrix(axis, \_objectPos, \_rotationSpeed).ExtractRotation());

\_camera.Position += \_objectPos;

\_camera.\_front = -Vector3.Normalize(\_camera.Position - \_objectPos);

\_objectPos /= 2;

}

// M (bawah -> Rotasi sumbu x)

if (KeyboardState.IsKeyDown(Keys.M))

{

\_objectPos \*= 2;

var axis = new Vector3(1, 0, 0);

\_camera.Position -= \_objectPos;

\_camera.Pitch += \_rotationSpeed;

\_camera.Position = Vector3.Transform(\_camera.Position,

generateArbRotationMatrix(axis, \_objectPos, -\_rotationSpeed).ExtractRotation());

\_camera.Position += \_objectPos;

\_camera.\_front = -Vector3.Normalize(\_camera.Position - \_objectPos);

\_objectPos /= 2;

}

// N (kiri -> Rotasi sumbu y)

if (KeyboardState.IsKeyDown(Keys.N))

{

\_objectPos \*= 2;

var axis = new Vector3(0, 1, 0);

\_camera.Position -= \_objectPos;

\_camera.Yaw += \_rotationSpeed;

\_camera.Position = Vector3.Transform(\_camera.Position,

generateArbRotationMatrix(axis, \_objectPos, \_rotationSpeed).ExtractRotation());

\_camera.Position += \_objectPos;

\_camera.\_front = -Vector3.Normalize(\_camera.Position - \_objectPos);

\_objectPos /= 2;

}

// , (kanan -> Rotasi sumbu y)

if (KeyboardState.IsKeyDown(Keys.Comma))

{

\_objectPos \*= 2;

var axis = new Vector3(0, 1, 0);

\_camera.Position -= \_objectPos;

\_camera.Yaw -= \_rotationSpeed;

\_camera.Position = Vector3.Transform(\_camera.Position,

generateArbRotationMatrix(axis, \_objectPos, -\_rotationSpeed).ExtractRotation());

\_camera.Position += \_objectPos;

\_camera.\_front = -Vector3.Normalize(\_camera.Position - \_objectPos);

\_objectPos /= 2;

}

// J (putar -> Rotasi sumbu z)

if (KeyboardState.IsKeyDown(Keys.J))

{

\_objectPos \*= 2;

var axis = new Vector3(0, 0, 1);

\_camera.Position -= \_objectPos;

\_camera.Position = Vector3.Transform(\_camera.Position,

generateArbRotationMatrix(axis, \_objectPos, \_rotationSpeed).ExtractRotation());

\_camera.Position += \_objectPos;

\_camera.\_front = -Vector3.Normalize(\_camera.Position - \_objectPos);

\_objectPos /= 2;

}

// L (putar -> Rotasi sumbu z)

if (KeyboardState.IsKeyDown(Keys.L))

{

\_objectPos \*= 2;

var axis = new Vector3(0, 0, 1);

\_camera.Position -= \_objectPos;

\_camera.Position = Vector3.Transform(\_camera.Position,

generateArbRotationMatrix(axis, \_objectPos, -\_rotationSpeed).ExtractRotation());

\_camera.Position += \_objectPos;

\_camera.\_front = -Vector3.Normalize(\_camera.Position - \_objectPos);

\_objectPos /= 2;

}

if (!IsFocused)

{

return;

}

const float sensitivity = 0.02f;

if (\_firstMove)

{

\_lastMousePosition = new Vector2(MouseState.X, MouseState.Y);

\_firstMove = false;

}

else

{

// Hitung selisih mouse position

var deltaX = MouseState.X - \_lastMousePosition.X;

var deltaY = MouseState.Y - \_lastMousePosition.Y;

\_lastMousePosition = new Vector2(MouseState.X, MouseState.Y);

\_camera.Yaw += deltaX \* sensitivity;

\_camera.Pitch -= deltaY \* sensitivity;

}

base.OnUpdateFrame(args);

}

}

}

**Mesh.cs**

using LearnOpenTK.Common;

using OpenTK.Windowing.Desktop;

using OpenTK.Mathematics;

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

using OpenTK.Graphics.OpenGL4;

namespace camera

{

class Mesh

{

protected List<Vector3> vertices = new List<Vector3>();

protected List<Vector2> texture = new List<Vector2>();

protected List<Vector3> normals = new List<Vector3>();

protected List<Vector3> vertices\_res = new List<Vector3>();

protected List<uint> vertexIndices = new List<uint>();

protected List<uint> textureIndices = new List<uint>();

protected List<uint> normalIndices = new List<uint>();

protected bool \_lightSource = false;

protected int \_VBO;

protected int \_VAO;

protected int \_EBO;

protected int \_NBO;

protected int \_TBO;

protected Shader \_shader;

protected Matrix4 \_transform;

protected Matrix4 \_transform\_tmp;

//lighting

//private readonly Vector3 \_lightPos = new Vector3(0f, 0f, 0f);

//private int \_vaoLamp;

//private Shader \_lampShader;

private Shader \_lightingShader;

private Texture \_diffuseMap;

private Texture \_specularMap;

protected Matrix4 \_projection;

protected Matrix4 \_view;

public Vector3 scaleratio = new Vector3(0.1f);

List<float> vertex = new List<float>();

public List<Mesh> child = new List<Mesh>();

public Mesh(bool lightSource = false)

{

\_lightSource = lightSource;

}

private List<Vector3> GenerateCircleVertices(float radius, int segments)

{

List<Vector3> vertices = new List<Vector3>();

float angleIncrement = (2 \* MathHelper.Pi) / segments;

for (int i = 0; i < segments; i++)

{

float angle = i \* angleIncrement;

float x = radius \* (float)Math.Cos(angle);

float y = radius \* (float)Math.Sin(angle);

vertices.Add(new Vector3(x, y, 0f));

}

return vertices;

}

public void setupObject(string p1, string diffuse, string specular, float Sizex, float Sizey)

{

\_transform = Matrix4.Identity;

LoadObjFile(p1);

for (int i = 0; i < vertexIndices.Count; i++)

{

int index = (int)vertexIndices[i];

vertex.Add(vertices[index].X);

vertex.Add(vertices[index].Y);

vertex.Add(vertices[index].Z);

index = (int)normalIndices[i];

vertex.Add(normals[index].X);

vertex.Add(normals[index].Y);

vertex.Add(normals[index].Z);

index = (int)textureIndices[i];

vertex.Add(texture[index].X);

vertex.Add(texture[index].Y);

}

\_VAO = GL.GenVertexArray();

GL.BindVertexArray(\_VAO);

\_VBO = GL.GenBuffer();

GL.BindBuffer(BufferTarget.ArrayBuffer, \_VBO);

GL.BufferData(BufferTarget.ArrayBuffer, vertex.Count \* sizeof(float), vertex.ToArray(), BufferUsageHint.StaticDraw);

\_lightingShader = new Shader("C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Shaders/shader.vert",

"C:/Users/tonia/Downloads/Camera\_revisi-20230621T085641Z-001/Camera\_revisi/Camera/Shaders/lighting.frag");

//\_lampShader = new Shader("C:/Users/kupan/source/repos/Camera/Camera/Shaders/shader.vert",

// "C:/Users/kupan/source/repos/Camera/Camera/Shaders/shader.frag");

var vertexLocation = \_lightingShader.GetAttribLocation("aPosition");

GL.EnableVertexAttribArray(vertexLocation);

GL.VertexAttribPointer(vertexLocation, 3, VertexAttribPointerType.Float, false, 8 \* sizeof(float), 0);

var normalLocation = \_lightingShader.GetAttribLocation("aNormal");

GL.EnableVertexAttribArray(normalLocation);

GL.VertexAttribPointer(normalLocation, 3, VertexAttribPointerType.Float, false, 8 \* sizeof(float), 3 \* sizeof(float));

var texCoordLocation = \_lightingShader.GetAttribLocation("aTexCoords");

GL.EnableVertexAttribArray(texCoordLocation);

GL.VertexAttribPointer(texCoordLocation, 2, VertexAttribPointerType.Float, false, 8 \* sizeof(float), 6 \* sizeof(float));

\_diffuseMap = Texture.LoadFromFile(diffuse);

\_specularMap = Texture.LoadFromFile(specular);

\_transform \*= Matrix4.CreateScale(scaleratio);

}

public void render(int mode, Camera \_camera,int dir)

{

//lighting

GL.BindVertexArray(\_VAO);

\_diffuseMap.Use(TextureUnit.Texture0);

\_specularMap.Use(TextureUnit.Texture1);

if (dir==1)

{

\_lightingShader.SetMatrix4("transform", \_transform);

\_lightingShader.SetMatrix4("view", \_camera.GetViewMatrix());

\_lightingShader.SetMatrix4("projection", \_camera.GetProjectionMatrix());

\_lightingShader.SetVector3("viewPos", \_camera.Position);

\_lightingShader.SetInt("material.diffuse", 0);

\_lightingShader.SetInt("material.specular", 1);

\_lightingShader.SetFloat("material.shininess", 64.0f);

\_lightingShader.SetVector3("light.direction", new Vector3(-0.1f, 0.7f, -2.2f));

\_lightingShader.SetVector3("light.ambient", new Vector3(0.7f));

\_lightingShader.SetVector3("light.diffuse", new Vector3(0.5f));

\_lightingShader.SetVector3("light.specular", new Vector3(0.5f, 0.5f, 0.5f));

\_lightingShader.Use();

}

else

{

\_lightingShader.SetMatrix4("transform", \_transform);

\_lightingShader.SetMatrix4("view", \_camera.GetViewMatrix());

\_lightingShader.SetMatrix4("projection", \_camera.GetProjectionMatrix());

\_lightingShader.SetVector3("viewPos", \_camera.Position);

\_lightingShader.SetInt("material.diffuse", 0);

\_lightingShader.SetInt("material.specular", 1);

\_lightingShader.SetFloat("material.shininess", 64.0f);

\_lightingShader.SetVector3("light.direction", new Vector3(0.1f, -0.7f, 2.2f));

\_lightingShader.SetVector3("light.ambient", new Vector3(0.7f));

\_lightingShader.SetVector3("light.diffuse", new Vector3(0.5f));

\_lightingShader.SetVector3("light.specular", new Vector3(0.5f, 0.5f, 0.5f));

\_lightingShader.Use();

}

switch (mode)

{

case 1:

GL.DrawArrays(PrimitiveType.Triangles, 0, vertex.Count / 8);

break;

case 2:

GL.DrawArrays(PrimitiveType.TriangleFan, 0, vertex.Count / 8);

break;

case 3:

GL.DrawArrays(PrimitiveType.LineStrip, 0, vertex.Count / 8);

break;

}

}

public void save()

{

\_transform\_tmp = \_transform;

}

public void reset()

{

\_transform = \_transform\_tmp;

}

public void rotate(float dr)

{

\_transform = \_transform \* Matrix4.CreateRotationY(MathHelper.DegreesToRadians(dr));

}

public void scale(float r)

{

\_transform = \_transform \* Matrix4.CreateScale(r);

}

public void translate(float dx, float dy, float dz)

{

\_transform = \_transform \* Matrix4.CreateTranslation(dx, dy, dz) ;

}

public Vector3 getPos()

{

return \_transform.ExtractTranslation();

}

public void LoadObjFile(string path)

{

if (!File.Exists(path))

{

throw new FileNotFoundException("Unable to open \"" + path + "\", does not exist.");

}

using (StreamReader streamReader = new StreamReader(path))

{

while (!streamReader.EndOfStream)

{

List<string> words = new List<string>(streamReader.ReadLine().ToLower().Split(' '));

words.RemoveAll(s => s == string.Empty);

if (words.Count == 0)

continue;

string type = words[0];

words.RemoveAt(0);

switch (type)

{

case "v":

vertices.Add(new Vector3(float.Parse(words[0]) / 10, float.Parse(words[1]) / 10, float.Parse(words[2]) / 10));

break;

case "vt":

texture.Add(new Vector2(float.Parse(words[0]), float.Parse(words[1])));

break;

case "vn":

normals.Add(new Vector3(float.Parse(words[0]), float.Parse(words[1]), float.Parse(words[2])));

break;

// face

case "f":

foreach (string w in words)

{

if (w.Length == 0)

continue;

string[] comps = w.Split('/');

// subtract 1: indices start from 1, not 0

vertexIndices.Add(uint.Parse(comps[0]) - 1);

textureIndices.Add(uint.Parse(comps[1]) - 1);

normalIndices.Add(uint.Parse(comps[2]) - 1);

}

break;

default:

break;

}

}

}

}

}

}

**Camera.cs**

using OpenTK.Mathematics;

using System;

namespace LearnOpenTK.Common

{

// This is the camera class as it could be set up after the tutorials on the website

// It is important to note there are a few ways you could have set up this camera, for example

// you could have also managed the player input inside the camera class, and a lot of the properties could have

// been made into functions.

// TL;DR: This is just one of many ways in which we could have set up the camera

// Check out the web version if you don't know why we are doing a specific thing or want to know more about the code

public class Camera

{

// Those vectors are directions pointing outwards from the camera to define how it rotated

public Vector3 \_front = -Vector3.UnitZ;

private Vector3 \_up = Vector3.UnitY;

private Vector3 \_right = Vector3.UnitX;

// Rotation around the X axis (radians)

private float \_pitch;

// Rotation around the Y axis (radians)

private float \_yaw = -MathHelper.PiOver2; // Without this you would be started rotated 90 degrees right

// The field of view of the camera (radians)

private float \_fov = MathHelper.PiOver2;

public Camera(Vector3 position, float aspectRatio)

{

Position = position;

AspectRatio = aspectRatio;

}

// The position of the camera

public Vector3 Position { get; set; }

// This is simply the aspect ratio of the viewport, used for the projection matrix

public float AspectRatio { private get; set; }

public Vector3 Front => \_front;

public Vector3 Up => \_up;

public Vector3 Right => \_right;

// We convert from degrees to radians as soon as the property is set to improve performance

public float Pitch

{

get => MathHelper.RadiansToDegrees(\_pitch);

set

{

// We clamp the pitch value between -89 and 89 to prevent the camera from going upside down, and a bunch

// of weird "bugs" when you are using euler angles for rotation.

// If you want to read more about this you can try researching a topic called gimbal lock

//var angle = MathHelper.Clamp(value, -89f, 89f);

\_pitch = MathHelper.DegreesToRadians(value);

UpdateVectors();

}

}

// We convert from degrees to radians as soon as the property is set to improve performance

public float Yaw

{

get => MathHelper.RadiansToDegrees(\_yaw);

set

{

\_yaw = MathHelper.DegreesToRadians(value);

UpdateVectors();

}

}

// The field of view (FOV) is the vertical angle of the camera view, this has been discussed more in depth in a

// previous tutorial, but in this tutorial you have also learned how we can use this to simulate a zoom feature.

// We convert from degrees to radians as soon as the property is set to improve performance

public float Fov

{

get => MathHelper.RadiansToDegrees(\_fov);

set

{

var angle = MathHelper.Clamp(value, 1f, 179f);

\_fov = MathHelper.DegreesToRadians(angle);

}

}

// Get the view matrix using the amazing LookAt function described more in depth on the web tutorials

public Matrix4 GetViewMatrix()

{

return Matrix4.LookAt(Position, Position + \_front, \_up);

}

// Get the projection matrix using the same method we have used up until this point

public Matrix4 GetProjectionMatrix()

{

return Matrix4.CreatePerspectiveFieldOfView(\_fov, AspectRatio, 0.01f, 100f);

}

// This function is going to update the direction vertices using some of the math learned in the web tutorials

private void UpdateVectors()

{

// First the front matrix is calculated using some basic trigonometry

\_front.X = MathF.Cos(\_pitch) \* MathF.Cos(\_yaw);

\_front.Y = MathF.Sin(\_pitch);

\_front.Z = MathF.Cos(\_pitch) \* MathF.Sin(\_yaw);

// We need to make sure the vectors are all normalized, as otherwise we would get some funky results

\_front = Vector3.Normalize(\_front);

// Calculate both the right and the up vector using cross product

// Note that we are calculating the right from the global up, this behaviour might

// not be what you need for all cameras so keep this in mind if you do not want a FPS camera

\_right = Vector3.Normalize(Vector3.Cross(\_front, Vector3.UnitY));

\_up = Vector3.Normalize(Vector3.Cross(\_right, \_front));

}

}

}