

Laporan Hasil Belajar Grafika Komputer
OpenTK & Shader



Dibuat Oleh :

Angga Kresnabayu	140810160001
Fauzi Faruq Nabbani	140810160007
Muhammad Jordiansyah	140810160040
Adryan Luthfi Faiz	140810160049
Muhammad Islam Taufikurahman	140810160062
Patricia Joanne	140810160065

UNIVERSITAS PADJADJARAN
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
PROGRAM STUDI S1 TEKNIK INFORMATIKA
2018

OpenTK

OpenTK, juga dikenal sebagai Library Open Toolkit, adalah Library grafis C # yang menggantikan Kerangka Tao. Ini menyediakan akses ke alat grafis yang terdapat di OpenGL, OpenCL, dan OpenAL ke berbagai bahasa berbasis CLR (C #, F #, dll.). Khususnya, ini dapat digunakan dengan bahasa apa pun yang menggunakan salah satu kerangka .Net atau Mono.

OpenTK dapat digunakan untuk permainan, aplikasi ilmiah atau proyek lain yang membutuhkan grafis 3D, audio atau fungsi komputasi.

Repositori resmi dapat ditemukan di github: <https://github.com/opentk/opentk>

Contoh Project :



Seberapa cepat OpenTK?

Untuk pustaka .Net, OpenTK sangat cepat. Ini menggunakan perakitan IL yang dioptimalkan untuk meminimalkan overhead ketika memanggil fungsi OpenGL

Shading Modelling

Shading mengacu pada penggambaran kedalaman suatu objek dalam model 3D atau ilustrasi dengan mengubah-ubah tingkat dari kegelapan suatu object(darkness).

Menggambar Shading merupakan suatu proses yang digunakan dalam menggambar dengan tingkat darkness tertentu pada sebuah kertas dengan memakai media yang lebih padat atau menampilkan bayangan yang lebih gelap untuk area yang lebih gelap dan memakai media yang tidak terlalu padat atau menampilkan bayangan yang lebih terang untuk area yang lebih terang. Ada berbagai macam teknik shading, misalnya cross hatching dimana garis-garis tegak lurus dengan jarak satu sama lain (kedekatan) yang berbeda-beda digambar pada pola grid untuk membentuk bayangan area. Semakin dekat garis-garis tersebut, semakin gelap area yang muncul. Begitu pula sebaliknya, semakin jauh garis-garis tersebut, semakin terang area yang muncul.

Pola-pola yang terang (misalnya objek yang memiliki area terang dan area berbayang) akan sangat membantu dalam pembuatan ilusi kedalaman pada kertas dan layar komputer.

Komputer grafis

Pada komputer grafis, shading mengacu pada proses mengubah warna berdasarkan sudut terhadap cahaya dan jarak dari cahaya untuk menciptakan efek photorealistic. Shading dilakukan selama proses penggambaran.

Model Shading

Model shading menentukan bagaimana suatu permukaan objek muncul dalam kondisi pencahayaan yang berbeda-beda. Beberapa model matematis dapat digunakan untuk menghitung shading. Setiap model shading memproses relasi dari permukaan normal terhadap sumber cahaya untuk menciptakan efek shading tertentu.

Phong



Menggunakan warna-warna ambient, diffuse, dan specular. Model shading ini membaca orientasi permukaan normal dan menginterpolasikannya untuk menciptakan tampilan smooth shading. Ia juga memproses relasi antara normal, cahaya, dan sudut pandang kamera untuk menciptakan specular highlight.

Hasilnya adalah suatu objek dengan bayangan smooth, permukaan area yang disinari diffuse dan ambient, serta suatu specular highlight sehingga objek tampak bersinar seperti bola biliar atau bola plastik. Pemantulan, transparansi, refraksi, dan tekstur dapat diterapkan pada objek yang menggunakan Phongshader.

Lambert



Menggunakan warna-warna ambient dan diffuse untuk menciptakan permukaan matte tanpa specular highlight. Ia menginterpolasikan normal dari permukaan segitiga yang berdampingan sehingga shading berubah secara progresif, menciptakan suatu permukaan matte.

Hasilnya adalah suatu objek dengan smooth shading, seperti telur atau bola ping-pong. Pemantulan, transparansi, refraksi, dan tekstur dapat diterapkan pada objek yang menggunakan Lambert shader.

Blinn



Menggunakan warna-warna diffuse, ambient, dan specular, serta refractive index untuk menghitung specular highlight. Model shading ini identik dengan model shading Phong, kecuali bentuk specular highlight-nya merefleksikan pencahayaan lebih akurat ketika ada sudut tinggi antara kamera dan cahaya.

Model shading ini berguna untuk tepian yang kasar atau tajam dan untuk mensimulasikan permukaan logam. Specular highlight-nya tampak lebih terang dibandingkan model Phong. Pemantulan, transparansi, refraksi, dan tekstur dapat diterapkan pada objek yang menggunakan Blinnshader.

Cook-Torrance



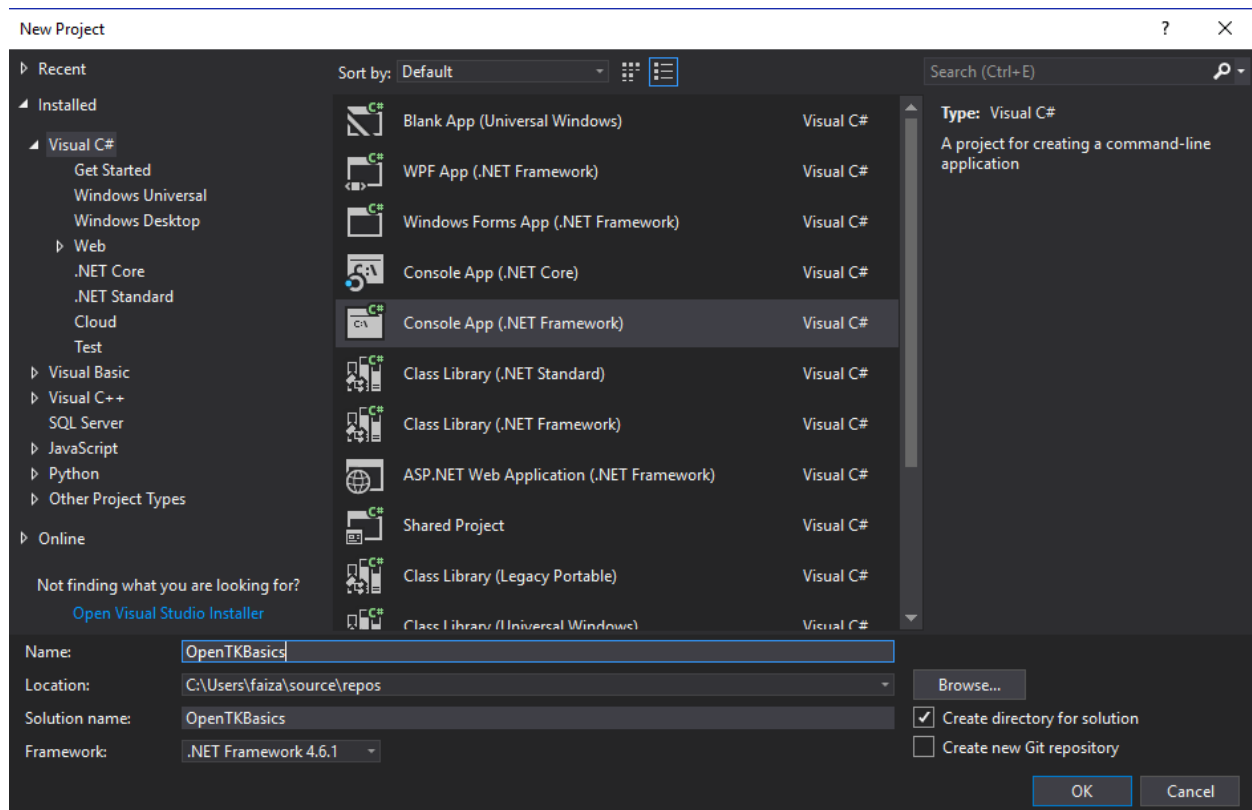
Menggunakan warna-warna diffuse, ambient, dan specular, serta refractive index untuk menghitung specular highlight. Ia membaca orientasi permukaan normal dan menginterpolasikannya untuk menciptakan tampilan smooth shading. Ia juga memproses relasi antara normal, cahaya, dan sudut pandang kamera untuk menciptakan specular highlight.

Model shading ini memproduksi hasil yang berada diantara model shading Blinn dan Lambert, serta berguna untuk mensimulasikan objek yang lembut dan reflektif seperti kulit. Pemantulan, transparansi, refraksi, dan tekstur dapat diterapkan pada objek yang menggunakan Cook-Torrance shader.

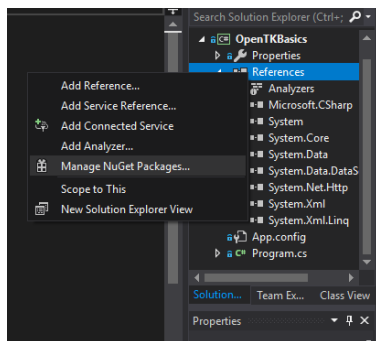
Karena model shading ini lebih kompleks untuk dihitung, ia memakan waktu lebih lama dalam pelukisan daripada model shading lainnya.

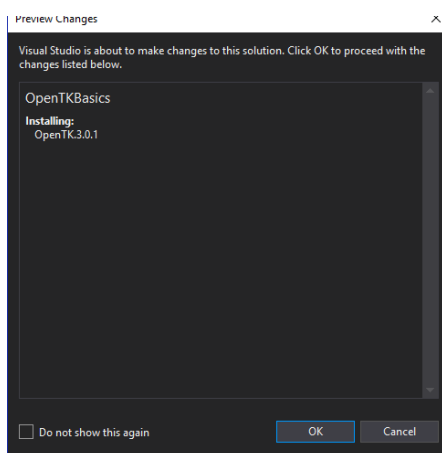
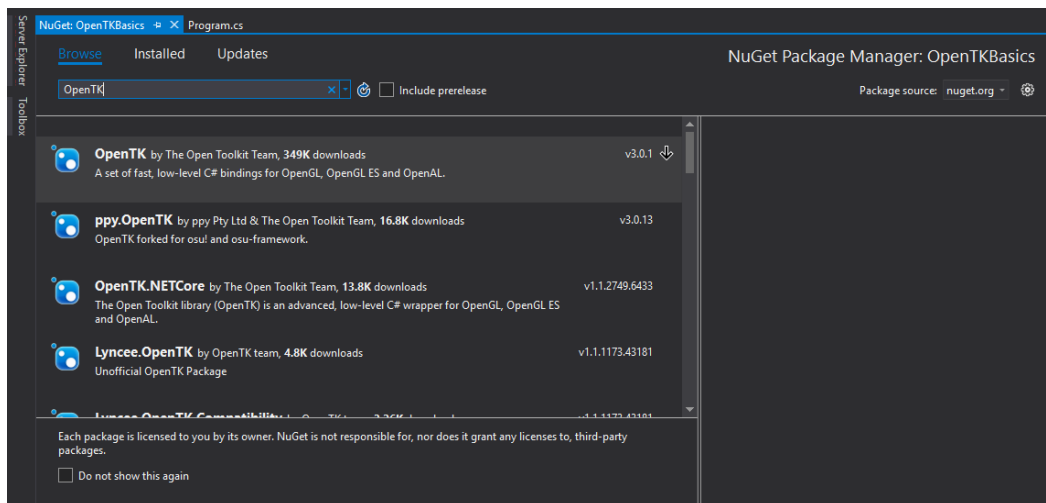
GameWindow and Setup (Implementasi OpenTK)

New Project >> Console App (.NET Framework)

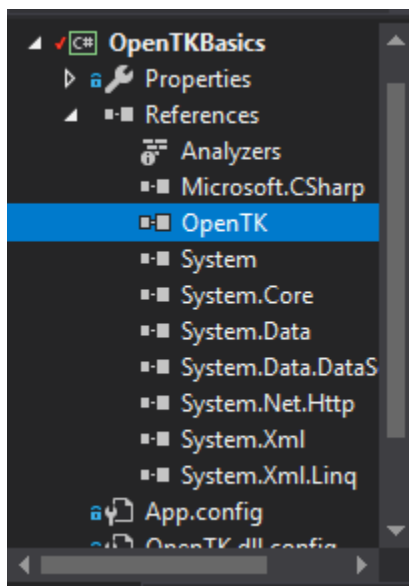


Install OpenTK dari Visual Studio

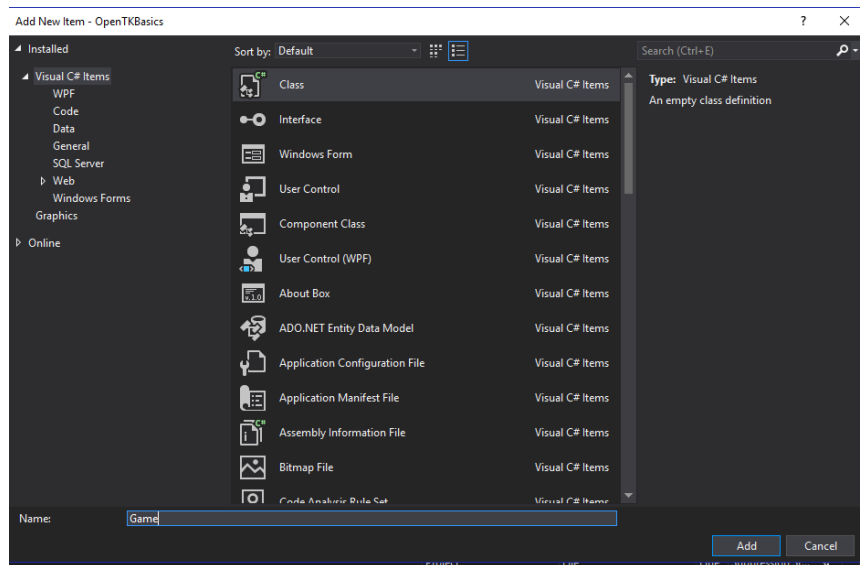
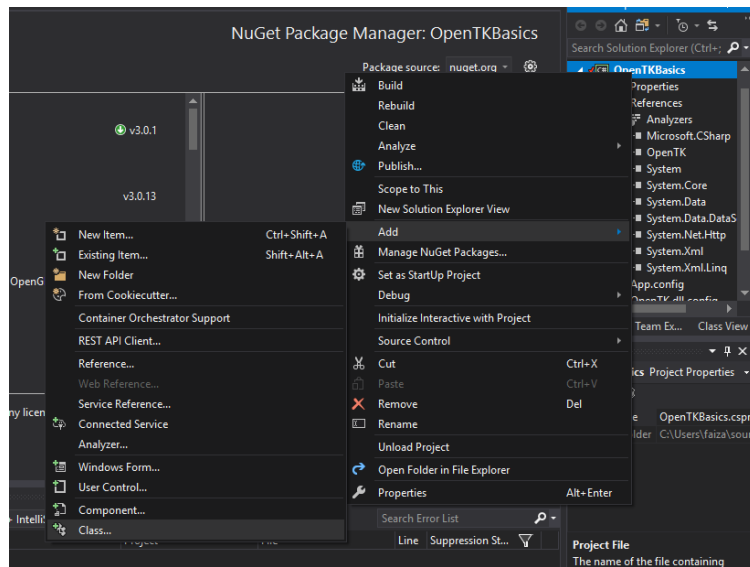


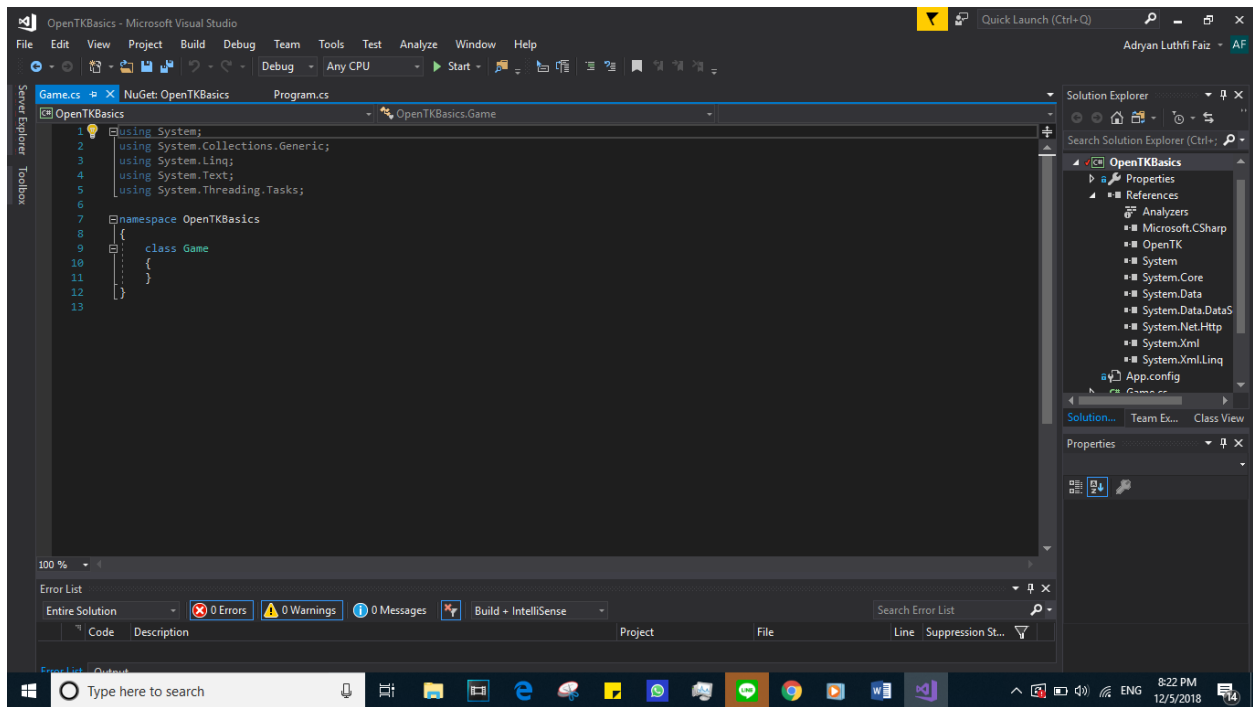


Setelah Terinstall



Buat Class





Masukkan (`using OpenTK;`)

```
using OpenTK;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

Coding di Class Game (Game.cs)

```
using OpenTK;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace OpenTKBasics
{
    class Game
    {
        public GameWindow window;

        public Game (GameWindow windowInput)
        {
            this.window = windowInput;
        }
    }
}
```

```
}
```

Coding di (program.cs)

```
using OpenTK;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

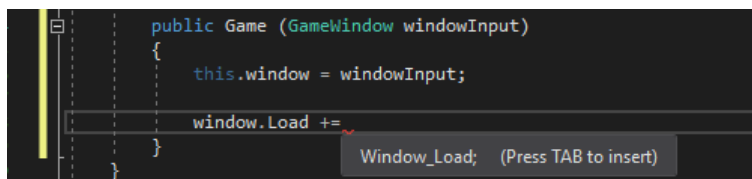
namespace OpenTKBasics
{
    class Program
    {
        static void Main(string[] args)
        {
            GameWindow window = new GameWindow(800, 600);
            Game game = new Game(window);

            window.Run();
        }
    }
}
```

Untuk menload eventhandler

Coding di game.cs

Tambahkan " window.Load += " dibawah "this.window = windowInput;"



```
public Game (GameWindow windowInput)
{
    this.window = windowInput;
    window.Load +=;
}
```

Window_Load; (Press TAB to insert)

Lalu press TAB

```

namespace OpenTKBasics
{
    class Game
    {
        public GameWindow window;

        public Game (GameWindow windowInput)
        {
            this.window = windowInput;

            window.Load += Window_Load;
        }

        private void Window_Load(object sender, EventArgs e)
        {
            throw new NotImplementedException();
        }
    }
}

```

Masukkan 3 kodingan ini

```

window.Load += Window_Load;
window.RenderFrame +=
window.UpdateFrame +=
window.Closing +=

```

Lakukan sama seperti yang sebelumnya

Press TAB

```

private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    throw new NotImplementedException();
}

private void Window_UpdateFrame(object sender, FrameEventArgs e)
{
    throw new NotImplementedException();
}

private void Window_RenderFrame(object sender, FrameEventArgs e)
{
    throw new NotImplementedException();
}

private void Window_Load(object sender, EventArgs e)
{
    throw new NotImplementedException();
}

```

Lalu dikosongkan

```
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    ...
}

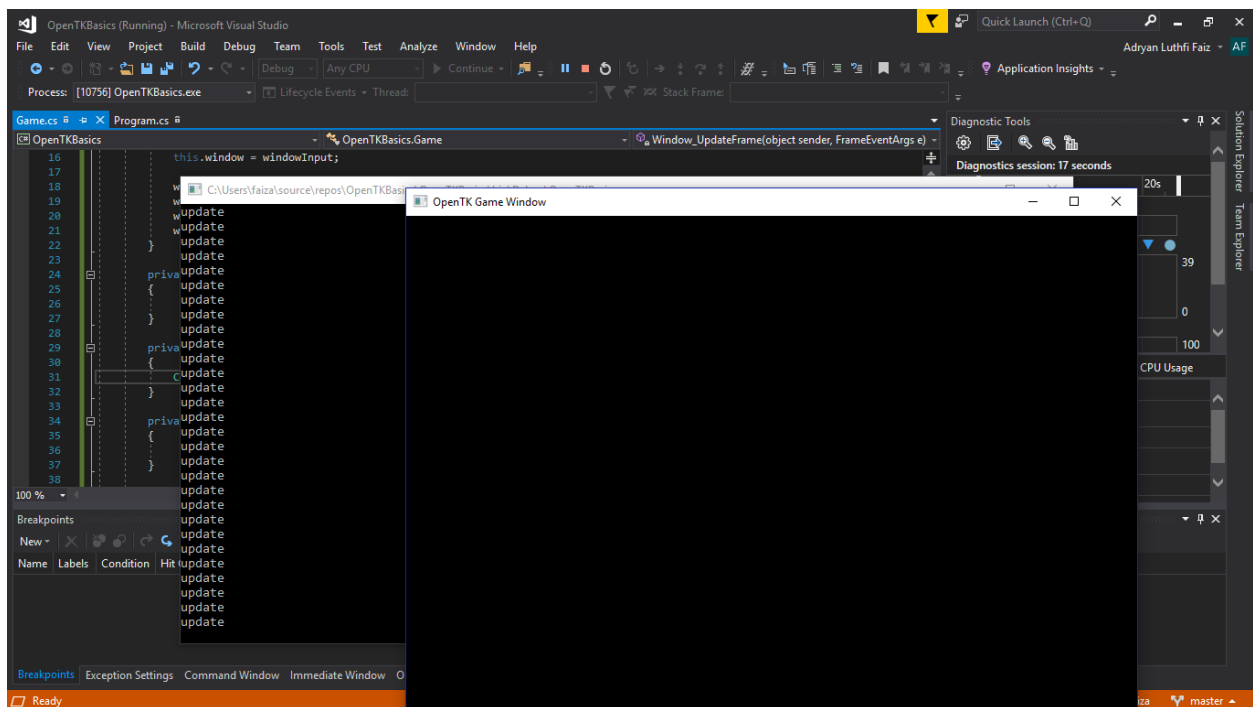
private void Window_UpdateFrame(object sender, FrameEventArgs e)
{
    ...
}

private void Window_RenderFrame(object sender, FrameEventArgs e)
{
    ...
}

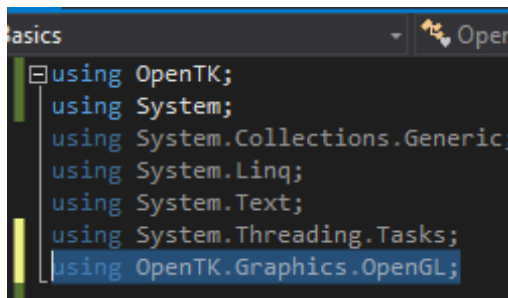
private void Window_Load(object sender, EventArgs e)
{
    ...
}
```

Mencoba Window_UpdateFrame

```
private void Window_UpdateFrame(object sender, FrameEventArgs e)
{
    Console.WriteLine("update");
}
```

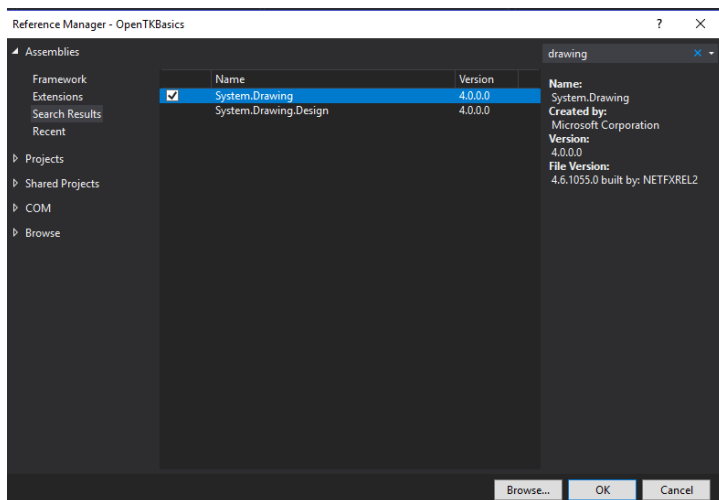
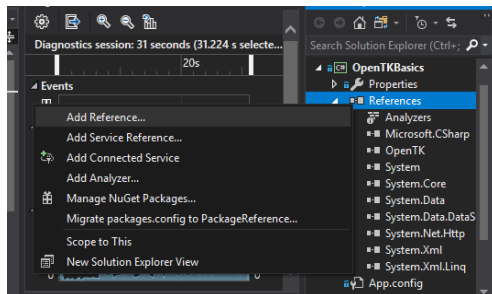


Tambahkan “`using OpenTK.Graphics.OpenGL;`”

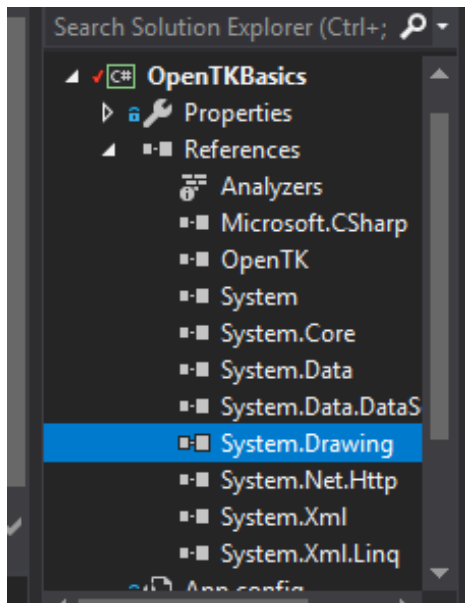


```
using OpenTK;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using OpenTK.Graphics.OpenGL;
```

Tambahkan References Color Drawing



Akan Bertambah



Panggil System.Drawing

```
using System.Drawing;
```

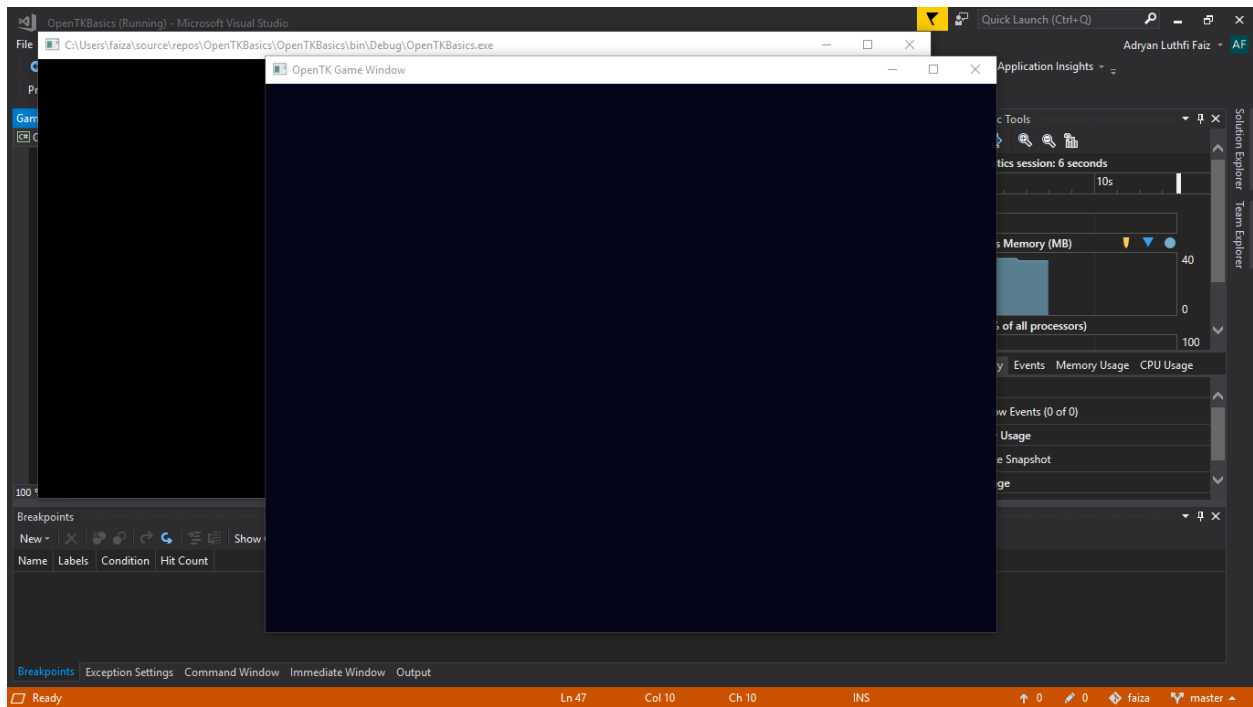
```
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
}

private void Window_UpdateFrame(object sender, FrameEventArgs e)
{
}

private void Window_RenderFrame(object sender, FrameEventArgs e)
{
    GL.Clear(ClearBufferMask.ColorBufferBit);

    GL.Flush();
    window.SwapBuffers();
}

private void Window_Load(object sender, EventArgs e)
{
    GL.ClearColor(Color.FromArgb(5, 5, 25));
}
```



Drawing in Immediate Mode (OpenTK)

```
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
}

private void Window_UpdateFrame(object sender, FrameEventArgs e)
{
}

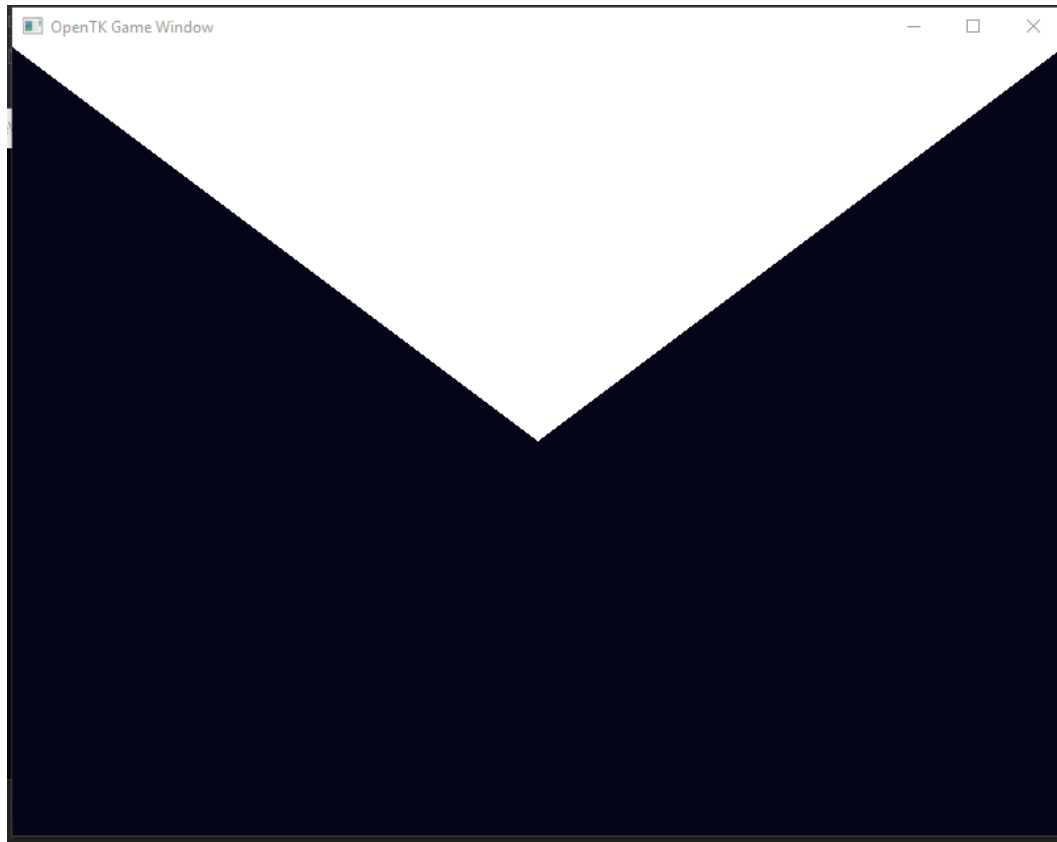
private void Window_Load(object sender, EventArgs e)
{
}

private void Window_RenderFrame(object sender, FrameEventArgs e)
{
    GL.ClearColor(Color.FromArgb(5, 5, 25));
    GL.Clear(ClearBufferMask.ColorBufferBit);

    GL.Begin(PrimitiveType.Triangles);
    GL.Vertex2(0, 0);
    GL.Vertex2(1, 1);
    GL.Vertex2(-1, 1);
    GL.End();

    GL.Flush();
}
```

```
} window.SwapBuffers();
```

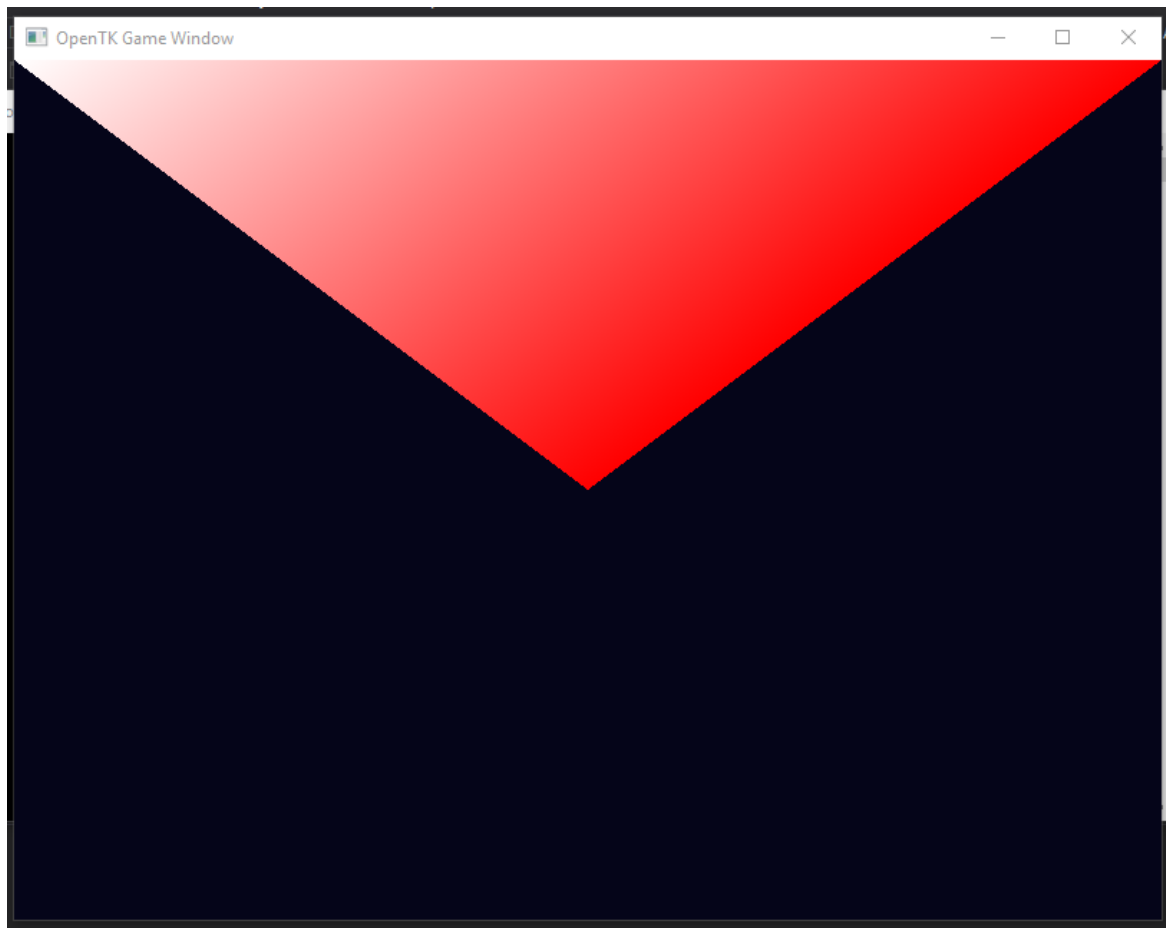


Tambahkan Color

```
private void Window_RenderFrame(object sender, FrameEventArgs e)
{
    GL.ClearColor(Color.FromArgb(5, 5, 25));
    GL.Clear(ClearBufferMask.ColorBufferBit);

    GL.Begin(PrimitiveType.Triangles);
    GL.Color3(Color.Red);
    GL.Vertex2(0, 0);
    GL.Vertex2(1, 1);
    GL.Color4(Color.FromArgb(128, 255, 255, 255));
    GL.Vertex2(-1, 1);
    GL.End();

    GL.Flush();
    window.SwapBuffers();
}
```

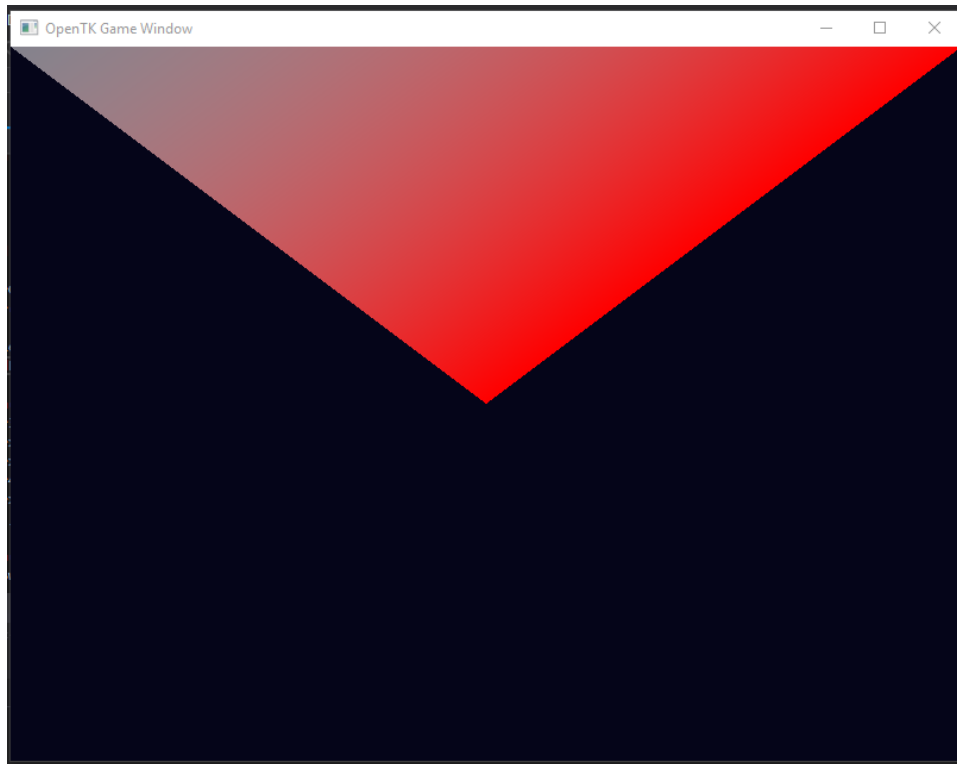
Tambahkan Blending

```
private void Window_RenderFrame(object sender, FrameEventArgs e)
{
    GL.ClearColor(Color.FromArgb(5, 5, 25));
    GL.Clear(ClearBufferMask.ColorBufferBit);

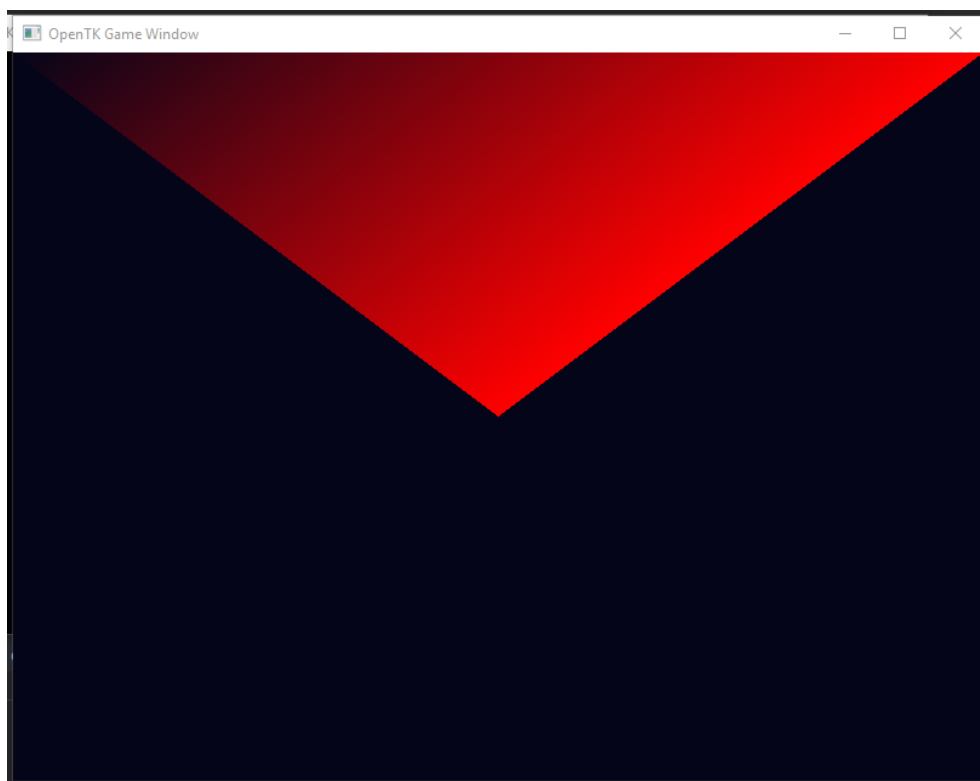
    GL.Enable(EnableCap.Blend);
    GL.BlendFunc(BlendingFactor.SrcAlpha, BlendingFactor.OneMinusSrcAlpha);

    GL.Begin(PrimitiveType.Triangles);
    GL.Color3(Color.Red);
    GL.Vertex2(0, 0);
    GL.Vertex2(1, 1);
    GL.Color4(Color.FromArgb(128, 255, 255, 255));
    GL.Vertex2(-1, 1);
    GL.End();

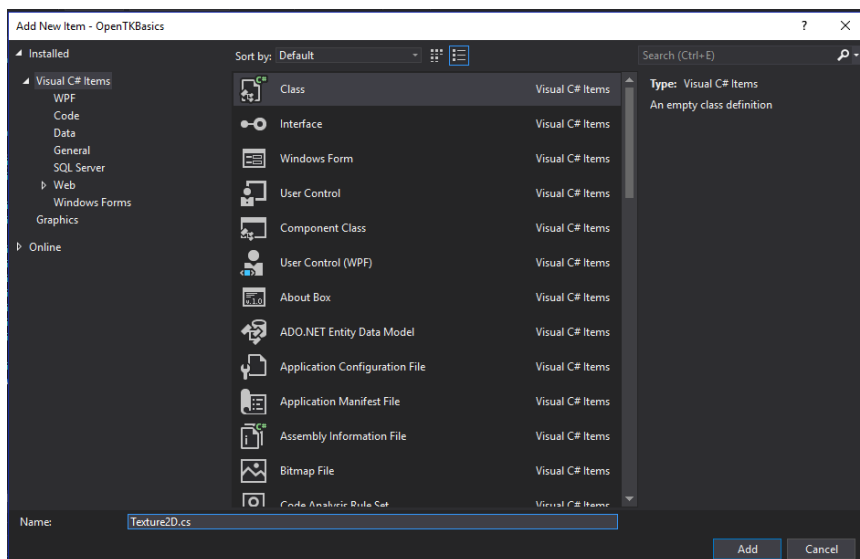
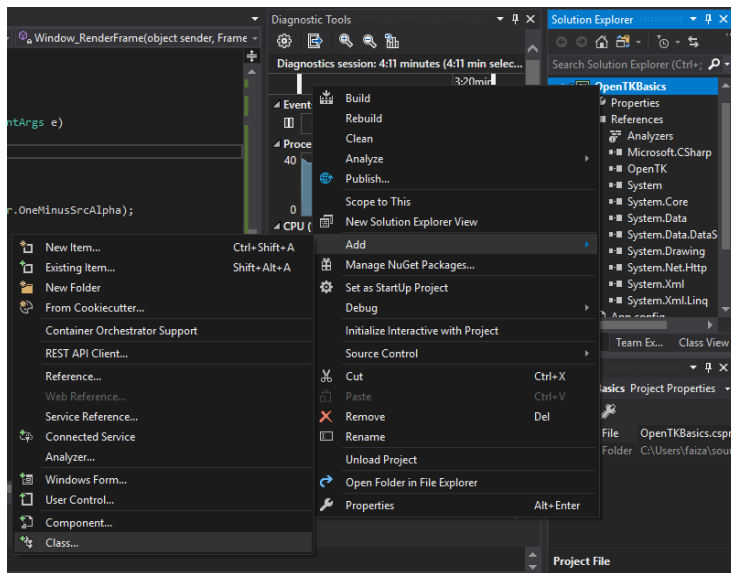
    GL.Flush();
    window.SwapBuffers();
}
```



```
GL.Color4(Color.FromArgb(0, Color.Red));
```



Bikin Class Texture2D



Coding di class Texture2D (Texture2D.cs)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using OpenTK;

namespace OpenTKBasics
{
    struct Texture2D
    {
        private int id;
        private Vector2 size;
    }
}
```

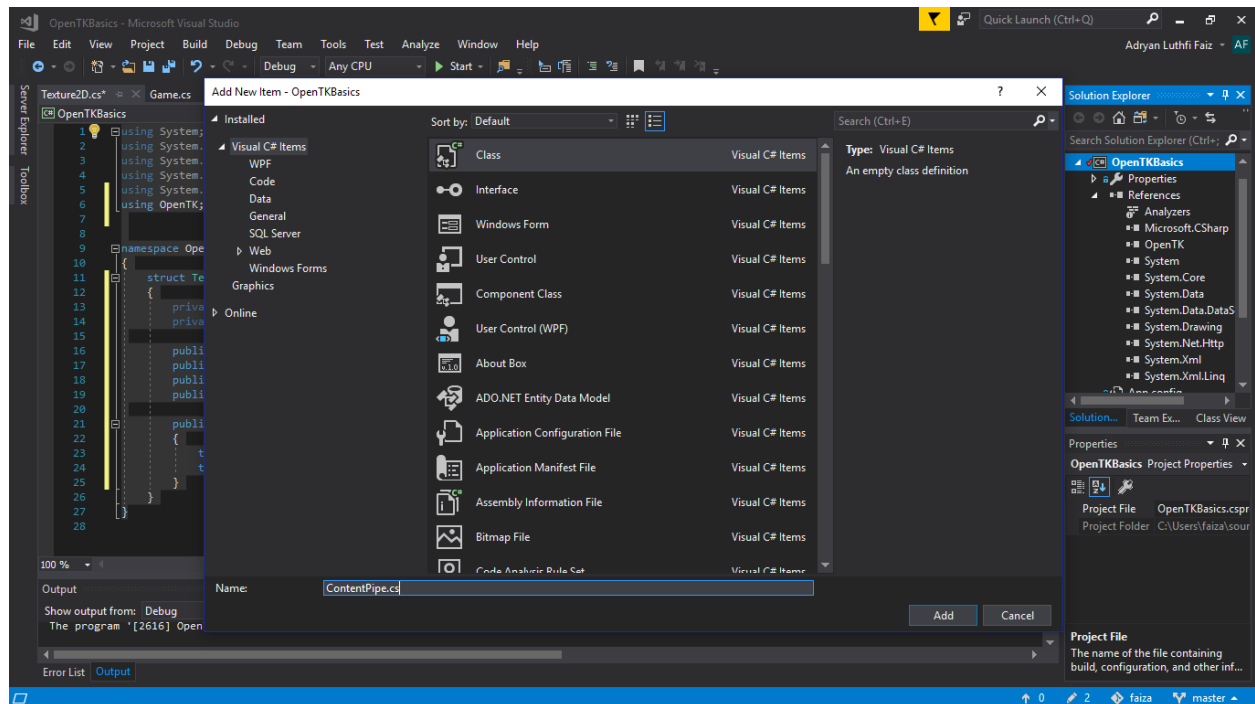
```

        public int ID { get { return id; } }
        public Vector2 Size { get { return size; } }
        public int Width { get { return (int)size.X; } }
        public int Height { get { return (int)size.Y; } }

        public Texture2D (int id, Vector2 size)
        {
            this.id = id;
            this.size = size;
        }
    }
}

```

Bikin Class ContentPipe

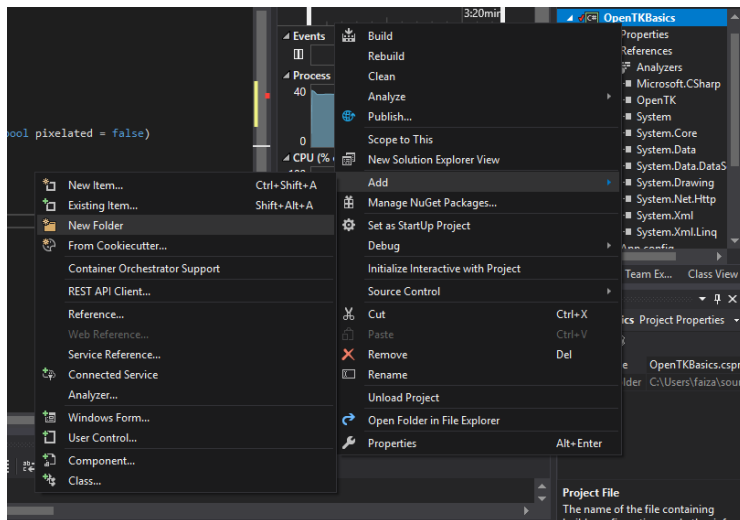


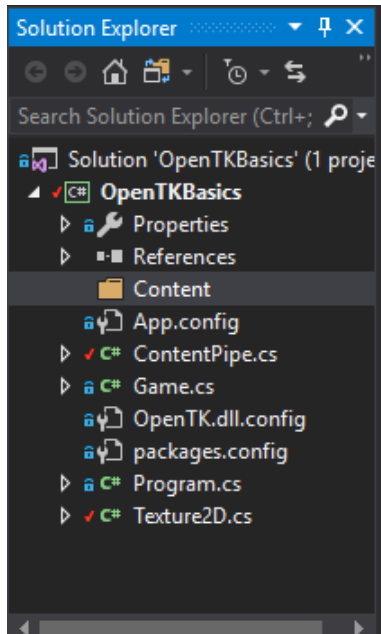
Coding di class ContentPipe (ContentPipe.cs)

```
ContentPipe.cs* X Texture2D.cs* Game.cs Program.cs
[CH] OpenTKBasics OpenTKBasics.ContentPipe LoadTexture(string filePath, bool pixelated = false)

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace OpenTKBasics
8 {
9     class ContentPipe
10    {
11        public static Texture2D LoadTexture(string filePath, bool pixelated = false)
12        {
13        }
14    }
15 }
16
17
```

Buat Folder Content





Masukkan `using System.IO; "`, `using System.Drawing; "`, `using System.Collections.Generic; "`, `using System.Drawing.Imaging; "`, `using OpenTK.Graphics.OpenGL; "`, `using OpenTK; "`

```
7 using System.IO;
8 using System.Drawing;
9 using System.Collections.Generic;
10 using System.Drawing.Imaging;
11 using OpenTK.Graphics.OpenGL;
12 using OpenTK;
13
```

Lanjutkan codingan di class Texture (Texture2D.cs)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.IO;
using System.Drawing;
using System.Collections.Generic;
using System.Drawing.Imaging;
using OpenTK.Graphics.OpenGL;
using OpenTK;

namespace OpenTKBasics
{
    class ContentPipe
    {
        public static Texture2D LoadTexture(string filePath, bool pixelated = false)
        {

```

```

        filePath = "Content/" + filePath;
        if (!File.Exists(filePath))
        {
            throw new Exception("File Does Not Exist At ' " + filePath + " ' ");
        }

        Bitmap bmp = new Bitmap(filePath);
        BitmapData data = bmp.LockBits(new Rectangle(0,0, bmp.Width, bmp.Height),
            ImageLockMode.ReadOnly,
            System.Drawing.Imaging.PixelFormat.Format32bppArgb);

        int id = GL.GenTexture();
        GL.BindTexture(TextureTarget.Texture2D, id);

        GL TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba,
        bmp.Width, bmp.Height, 0,
            OpenTK.Graphics.OpenGL.PixelFormat.Bgra, PixelType.UnsignedByte,
            data.Scan0);

        bmp.UnlockBits(data);

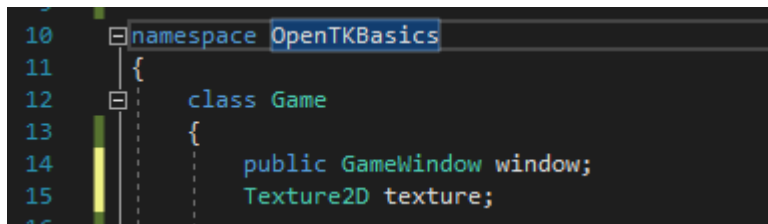
        GL TexParameter(TextureTarget.Texture2D,
        TextureParameterName.TextureMinFilter,
            pixelated ? (int)TextureMinFilter.Nearest :
            (int)TextureMinFilter.Linear);
        GL TexParameter(TextureTarget.Texture2D,
        TextureParameterName.TextureMagFilter,
            pixelated ? (int)TextureMagFilter.Nearest : (int)TextureMagFilter.Linear);

        return new Texture2D(id, new Vector2(bmp.Width, bmp.Height));
    }
}

```

Coding di class Game (Game.cs)

Tambahkan "Texture2D texture;"



```

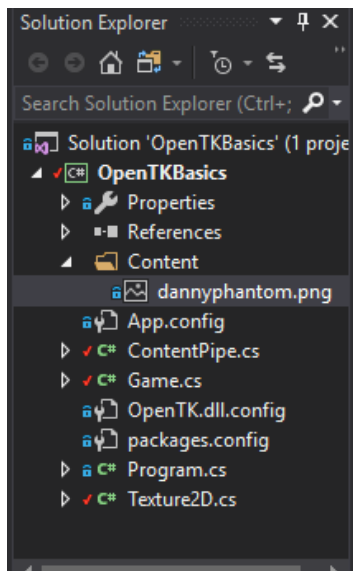
10 namespace OpenTKBasics
11 {
12     class Game
13     {
14         public GameWindow window;
15         Texture2D texture;
16     }
17 }

```

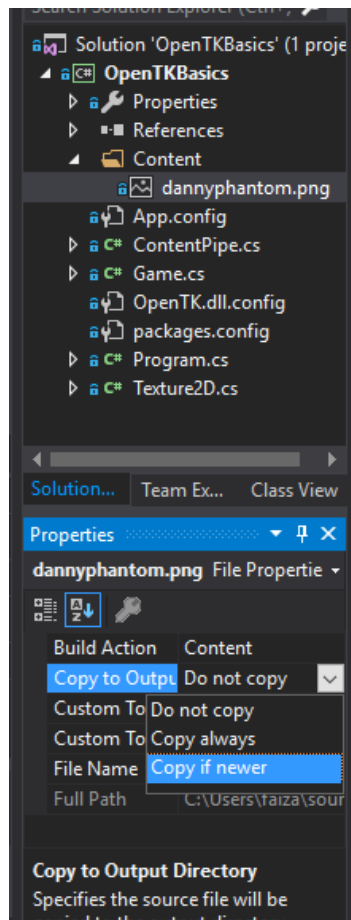
Masukkan Gambar ke dalam folder "Content"



dannyphantom.png



Ubah (Copy To Output) menjadi “ Copy Of Newer”



Coding pada bagian “ Window_Load ”

```
private void Window_Load(object sender, EventArgs e)
{
    texture = ContentPipe.LoadTexture("dannyphantom.png");
}
```

Coding pada bagian “Window_RenderFrame ”

Tambahkan

“ GL.BindTexture(TextureTarget.Texture2D, texture.ID); “,

“ GL.Enable(EnableCap.Texture2D); “

“ GL.TexCoord2(<x>, <y>); “

```

private void Window_RenderFrame(object sender, FrameEventArgs e)
{
    GL.ClearColor(Color.FromArgb(5, 5, 25));
    GL.Clear(ClearBufferMask.ColorBufferBit);

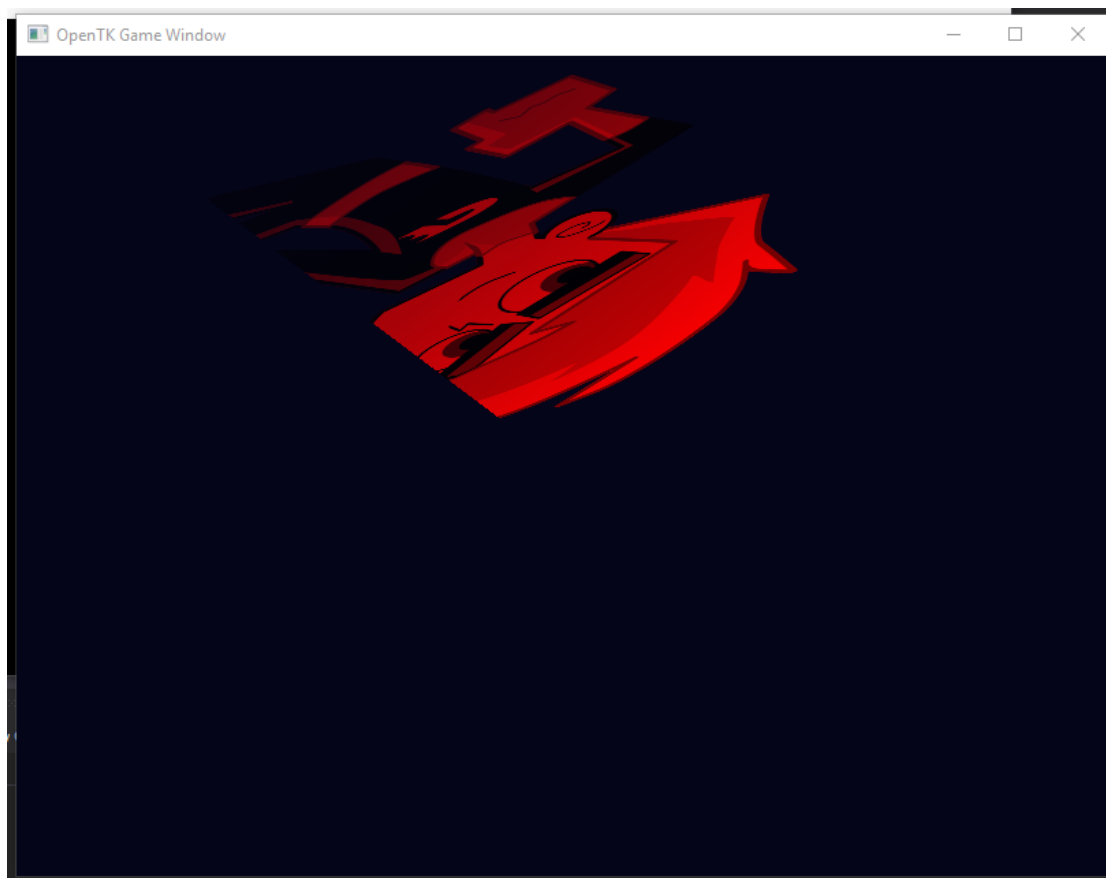
    GL.Enable(EnableCap.Texture2D);
    GL.Enable(EnableCap.Blend);
    GL.BlendFunc(BlendingFactor.SrcAlpha, BlendingFactor.OneMinusSrcAlpha);

    GL.BindTexture(TextureTarget.Texture2D, texture.ID);

    GL.Begin(PrimitiveType.Triangles);
    GL.Color3(Color.Red);
    GL.TexCoord2(0, 0); GL.Vertex2(0, 0);
    GL.TexCoord2(1, 0); GL.Vertex2(1, 1);
    GL.Color4(Color.FromArgb(0, Color.Red));
    GL.TexCoord2(1, 1); GL.Vertex2(-1, 1);
    GL.End();

    GL.Flush();
    window.SwapBuffers();
}

```



Contoh Shader

Source Code

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>

#include <iostream>
#include <cmath>

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void processInput(GLFWwindow *window);

// settings
const unsigned int SCR_WIDTH = 800;
const unsigned int SCR_HEIGHT = 600;

const char *vertexShaderSource = "#version 330 core\n"
    "layout (location = 0) in vec3 aPos;\n"
    "void main()\n"
    "{\n"
    "    gl_Position = vec4(aPos, 1.0);\n"
    "}\n0";

const char *fragmentShaderSource = "#version 330 core\n"
    "out vec4 FragColor;\n"
    "uniform vec4 ourColor;\n"
    "void main()\n"
    "{\n"
    "    FragColor = ourColor;\n"
    "}\n\n0";

int main()
{
    // glfw: initialize and configure
    // -----
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE); // uncomment
    this statement to fix compilation on OS X
#endif
}
```

```

// glfw window creation
// -----
GLFWwindow* window = glfwCreateWindow(800, 600, "LearnOpenGL",
NULL, NULL);
if (window == NULL)
{
    std::cout << "Failed to create GLFW window" << std::endl;
    glfwTerminate();
    return -1;
}
glfwMakeContextCurrent(window);
glfwSetFramebufferSizeCallback(window,
framebuffer_size_callback);

// glad: load all OpenGL function pointers
// -----
if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
{
    std::cout << "Failed to initialize GLAD" << std::endl;
    return -1;
}

// build and compile our shader program
// -----
// vertex shader
int vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
glCompileShader(vertexShader);
// check for shader compile errors
int success;
char infoLog[512];
glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
if (!success)
{
    glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);
    std::cout << "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n"
<< infoLog << std::endl;
}
// fragment shader
int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
glCompileShader(fragmentShader);
// check for shader compile errors
glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &success);
if (!success)
{
    glGetShaderInfoLog(fragmentShader, 512, NULL, infoLog);

```

```

        std::cout <<
"ERROR::SHADER::FRAGMENT::COMPILATION_FAILED\n" << infoLog <<
std::endl;
    }
    // link shaders
    int shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);
    // check for linking errors
    glGetProgramiv(shaderProgram, GL_LINK_STATUS, &success);
    if (!success) {
        glGetProgramInfoLog(shaderProgram, 512, NULL, infoLog);
        std::cout << "ERROR::SHADER::PROGRAM::LINKING_FAILED\n" <<
infoLog << std::endl;
    }
    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);

    // set up vertex data (and buffer(s)) and configure vertex
attributes
    // -----
    -----
    float vertices[] = {
        0.5f, -0.5f, 0.0f, // bottom right
        -0.5f, -0.5f, 0.0f, // bottom left
        0.0f, 0.5f, 0.0f // top
    };

    unsigned int VBO, VAO;
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    // bind the Vertex Array Object first, then bind and set vertex
buffer(s), and then configure vertex attributes(s).
    glBindVertexArray(VAO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);

    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
    glEnableVertexAttribArray(0);

    // You can unbind the VAO afterwards so other VAO calls won't
accidentally modify this VAO, but this rarely happens. Modifying
other

```

```

    // VAOs requires a call to glBindVertexArray anyways so we generally
    don't unbind VAOs (nor VBOs) when it's not directly necessary.
    // glBindVertexArray(0);

    // bind the VAO (it was already bound, but just to
    demonstrate): seeing as we only have a single VAO we can
    // just bind it beforehand before rendering the respective
    triangle; this is another approach.
    glBindVertexArray(VAO);

    // render loop
    // -----
    while (!glfwWindowShouldClose(window))
    {
        // input
        // ----
        processInput(window);

        // render
        // -----
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        // be sure to activate the shader before any calls to
        glUniform          glUseProgram(shaderProgram);

        // update shader uniform
        float timeValue = glfwGetTime();
        float greenValue = sin(timeValue) / 2.0f + 0.5f;
        int vertexColorLocation = glGetUniformLocation(shaderProgram,
"ourColor");
        glUniform4f(vertexColorLocation, 0.0f, greenValue, 0.0f,
1.0f);

        // render the triangle
        glDrawArrays(GL_TRIANGLES, 0, 3);

        // glfw: swap buffers and poll IO events (keys
        pressed/released, mouse moved etc.)
        // -----
        -----
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

```

```

    // optional: de-allocate all resources once they've outlived
    their purpose:
    // -----
    -----
    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);

    // glfw: terminate, clearing all previously allocated GLFW
    resources.
    // -----
    -----
    glfwTerminate();
    return 0;
}

// process all input: query GLFW whether relevant keys are
// pressed/released this frame and react accordingly
// -----
-----

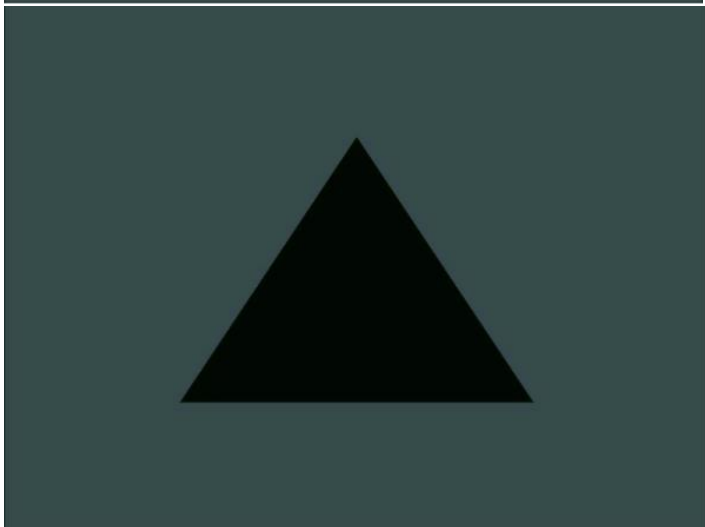
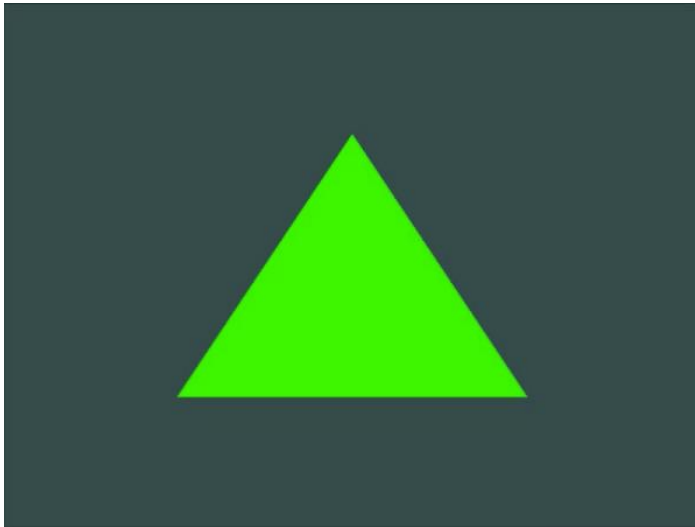
void processInput(GLFWwindow *window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}

// glfw: whenever the window size changed (by OS or user resize)
// this callback function executes
// -----
-----

void framebuffer_size_callback(GLFWwindow* window, int width, int
height)
{
    // make sure the viewport matches the new window dimensions;
    note that width and
    // height will be significantly larger than specified on retina
    displays.
    glViewport(0, 0, width, height);
}

```

Hasil Codingan



Referensi

<https://faris6593.blogspot.com/2014/04/konsep-dan-teori-pada-shading-modelling.html>

<https://sourceforge.net/projects/opentk/>

<https://en.wikipedia.org/wiki/OpenTK>

<https://opentk.net/faq.html>

<https://learnopengl.com/Getting-started/Shader>

https://learnopengl.com/code_viewer_gh.php?code=src/1.getting_started/3.1.shaders_uniform_shader/s_uniform.cpp

<https://youtu.be/8yHiyBQr-kk>

<https://youtu.be/Q23Kf9QEaO4>