ЛАБОРАТОРНА РОБОТИ OpenGL

Мета роботи: знайомлення з функціоналом графічної бібліотеки

Хід роботи:

1. Створимо вікно яке міняє колір за допомогою OpenGl

Program.cs

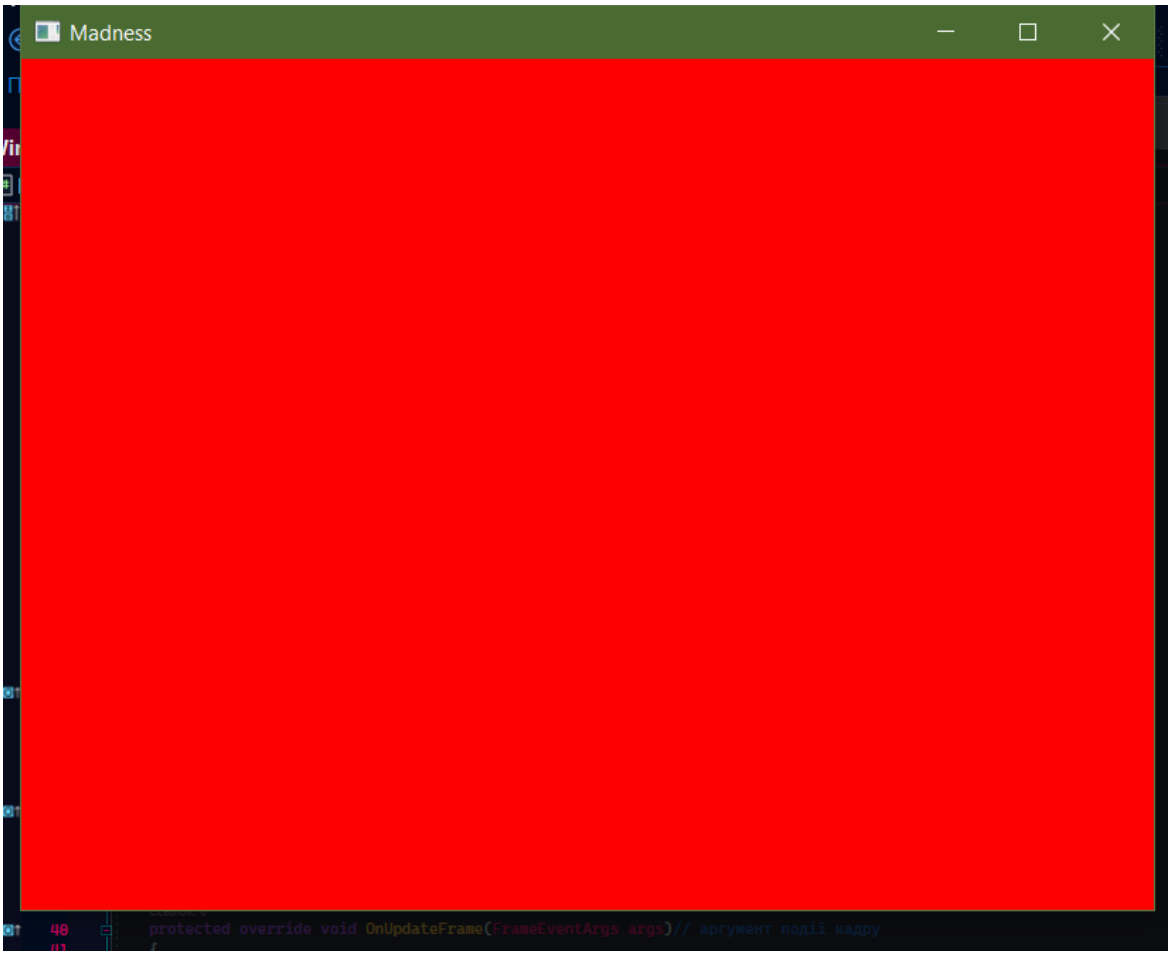
```
using OpenTK.Windowing.Desktop;
using OpenTK.Mathematics;

internal class Program
{
   private static void Main(string[] args)
   {
      //налаштуємо вікно
      var nativeWindowSettings = new NativeWindowSettings() {
         // елементарні налаштування вікна
         Size = new Vector2i(800, 600),
         Title = "Madness",

      };

      //створимо екземпляр вікна
      using(var window = new Window(GameWindowSettings.Default,
nativeWindowSettings))
      {
         window.Run();
      }
   }
}
```

Window.cs

```
using OpenTK.Graphics.OpenGL;
using OpenTK.Windowing.Common;
using OpenTK.Windowing.Desktop;
using OpenTK.Windowing.GraphicsLibraryFramework;

//Тут буде написаний увесь код OpenGL
//Open Toolkit дозволяє перевизначати кілька функцій для розширення
функціональності;
```

```
public class Window : GameWindow
{
  // Простий конструктор, який дозволяє установлювати такі властивості, як розмір вікна, заголовок, FPS
    public Window(GameWindowSettings gameWindowSettings, NativeWindowSettings nativeWindowSettings)
        : base(gameWindowSettings, nativeWindowSettings)
    {
    }
    /* ігровий цикл:
         Initialisation
             |
          __isRun___
         |   |    |
         |__Resize  |
         |   |    |
         |__Update  |
         |   |    |
         |__Draw    |
                |
             _____|
             |
           Delete*/
    //перевизначимо метод ініціалізації
    protected override void OnLoad()
    {
        base.OnLoad();
    }
    //метод зміни розмірів
    protected override void OnResize(ResizeEventArgs e) //аргумент події зміни розмірів
    {
        base.OnResize(e);
    }
    //метод оновлення кадру
    protected override void OnUpdateFrame(FrameEventArgs args)// аргумент події кадру
    {
        //назначимо r g b на зміну кольорів
        if (KeyboardState.IsKeyDown(Keys.R))
            GL.ClearColor(1.0f, 0.0f, 0.0f, 1.0f);
        if (KeyboardState.IsKeyDown(Keys.G))
            GL.ClearColor(0.0f, 1.0f, 0.0f, 1.0f);
        if (KeyboardState.IsKeyDown(Keys.B))
            GL.ClearColor(0.0f, 0.0f, 1.0f, 1.0f);
        //назначимо ескей на вимкнення
        if (KeyboardState.IsKeyDown(Keys.Escape))
            Close();
        base.OnUpdateFrame(args);
    }
    //метод відмальовки кадру
    protected override void OnRenderFrame(FrameEventArgs args)
    {
        //застосуємо очищення екрану кольором
```

```
            GL.Clear(ClearBufferMask.ColorBufferBit);
            //зміна буферу
            SwapBuffers();
            base.OnRenderFrame(args);
        }
        //метод обробника закриття вікна
        protected override void OnUnload()
        {
            base.OnUnload();
        }
    }
```



2.Відмалюємо примітив:

shader.vert

```
// встановлюємо версію GLSL
#version 330 core

//встановимо вхідну змінну aPosition
in vec3 aPosition;


void main(void)
{
// gl_Position – кінцева позиція вершини;
    gl_Position = vec4(aPosition, 1.0);
      }
```

shader.frag

```
#version 330 core

out vec4 outputColor;

void main()
{
    outputColor = vec4(1.0, 1.0, 0.0, 1.0);
      }
```

Window.cs

```
//побудуємо примітив
public class Window : GameWindow
{

    // побудуємо вершини трикутника
    private readonly float[] _vertices =
    {
         0.0f,  0.5f, 0.0f, //верхня вершина
        -0,5f, -0.5f, 0.0f, //нижня ліва вершина
         0,5f, -0,5f, 0.0f, //нижня права вершина
    };

    // оголосимо дескриптори VBA і VAO
    private int _vertexBufferObject;

    private int _vertexArrayObject;

    // створимо об'єкт шейдеру
    private Shader _shader;

    public Window(GameWindowSettings gameWindowSettings, NativeWindowSettings nativeWin-
dowSettings)
        : base(gameWindowSettings, nativeWindowSettings)
    {
    }

    protected override void OnLoad()
    {
        base.OnLoad();

        // встановимо колір очищення
        GL.ClearColor(0.34f, 0.47f, 0.2f, 1.0f);

        // Нам потрібно обробляти вершини графічною картою
        // створимо VBO

        // спочатку створимо буфер. ця функція повертає дескриптор
        _vertexBufferObject = GL.GenBuffer();

        // прив'яжемо буфер
        // Перший аргумент – це enum, що визначає тип буфера, який ми прив’язуємо. VBO –
це ArrayBuffer.
        // Другим аргументом є дескриптор нашого буфера.
        GL.BindBuffer(BufferTarget.ArrayBuffer, _vertexBufferObject);

        // тепер завантажимо вершини в буфер
        // Аргументи функції:
        //      1.До якого буфера надсилати дані
        //      2.Обсяг даних у байтах
        //      3.Самі вершини
        //      4.Спосіб використання буферу
```

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

        // оскільки буфер вершин не структорований і це просто купа даних
        // структуруємо його через VAO
        // створимо VAO
        _vertexArrayObject = GL.GenVertexArray();
        GL.BindVertexArray(_vertexArrayObject);

        // Тепер нам потрібно налаштувати, як вершинний шейдер інтерпретуватиме дані VBO;
        // Для цього ми використовуємо функцію GL.VertexAttribPointer
        // Ця функція має два завдання: повідомити opengl про формат даних, а також
пов'язати поточний буфер масиву з VAO.
        // Аргументи:
        //      1.Розташування вхідної змінної в шейдері
        //      2.Скільки елементів буде надіслано до змінної
        //      3.Тип набору елементів
        //      4.Чи потрібно перетворювати дані в нормалізовані координати
        //      5.Крок
        //      6.Зсув
        //      7.Stride і Offset
        GL.VertexAttribPointer(0, 3, VertexAttribPointerType.Float, false, 3 *
sizeof(float), 0);

        //увімкнемо 0 атрибут шейдеру
        GL.EnableVertexAttribArray(0);

        // тепер створимо шейдерні програми
        _shader = new Shader("Shaders/shader.vert", "Shaders/shader.frag");

        //тепер увімкнемо шейдер
        _shader.UseShader();

        //налаштування завершено перейдемо до відмальовки трикутника

    }
    protected override void OnResize(ResizeEventArgs e)
    {
        base.OnResize(e);
        // відкоректуємо відображення при зміні розмірів екрану
        GL.Viewport(0, 0, Size.X, Size.Y);
    }
    protected override void OnUpdateFrame(FrameEventArgs args)
    {
        if (KeyboardState.IsKeyDown(Keys.Escape))
            Close();
        base.OnUpdateFrame(args);
    }

    // Тепер, коли ініціалізацію виконано, створимо цикл візуалізації
    protected override void OnRenderFrame(FrameEventArgs args)
    {
        //очистимо зображення використовуючи GL.ClearColor який ми встановили раніше
        //для цього очистимо бітовий буфер кольору
        GL.Clear(ClearBufferMask.ColorBufferBit);

        // біндим шейдер
        _shader.UseShader();

        // підв'язуємо VAO
        GL.BindVertexArray(_vertexArrayObject);

        // виключемо функцію малювання
        GL.DrawArrays(PrimitiveType.Triangles, 0, 3);
```

```
        // це все що потрібно для відмальовки
        SwapBuffers();
        base.OnRenderFrame(args);
    }

    //загалом не слід видаляти все при виході з програми але хай буде поки так
    protected override void OnUnload()
    {
        GL.BindBuffer(BufferTarget.ArrayBuffer, 0);
        GL.BindVertexArray(0);
        GL.UseProgram(0);

        GL.DeleteBuffer(_vertexBufferObject);
        GL.DeleteVertexArray(_vertexArrayObject);

        GL.DeleteProgram(_shader.Handle);

        base.OnUnload();
    }
      }
```

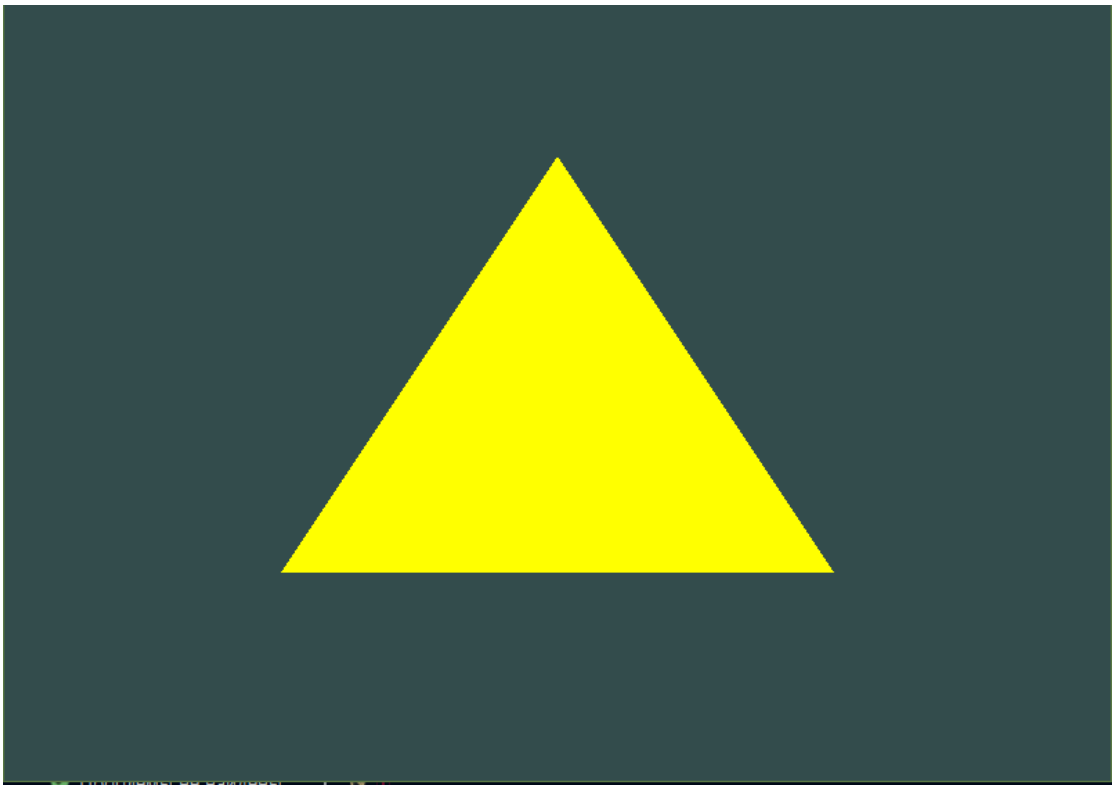
Shader.cs

```
//реалізуємо клас для створення шейдерів
public sealed class Shader
{
    public readonly int Handle;

    private readonly Dictionary<string, int> _uniformLocations;

    public Shader(string vertPath, string fragPath)
    {
        // Завантажуємо вершинний шейдер і компілюємо
        var shaderSource = File.ReadAllText(vertPath);

        // Створимо порожній вершинний шейдер
        var vertexShader = GL.CreateShader(ShaderType.VertexShader);// Перелік ShaderType
вказує, який тип шейдера буде створено

        // Тепер зв'яжемо код GLSL
        GL.ShaderSource(vertexShader, shaderSource);

        // А тепер скомпілюємо його
        CompileShader(vertexShader);

        // зробимо те саме і для фрагментного шейдеру
        shaderSource = File.ReadAllText(fragPath);
        var fragmentShader = GL.CreateShader(ShaderType.FragmentShader);
        GL.ShaderSource(fragmentShader, shaderSource);
        CompileShader(fragmentShader);

        // тепер об'єднаємо ці два шейдера в шейдерну програму
        Handle = GL.CreateProgram();//функція ініціалізує програму і повертає обробник

        // робимо прив'язку обох шейдерів
        GL.AttachShader(Handle, vertexShader);
        GL.AttachShader (Handle, fragmentShader);

        // тепер зв'яжемо їх
        LinkProgram(Handle);

        // оскільки ми вже прив'язали шейдери до програми то нам не потрібні окремі шей-
дери
        // тому відв'яжемо і видалимо їх
        GL.DetachShader(Handle, fragmentShader);
        GL.DetachShader(Handle, vertexShader);
```

```
        GL.DeleteShader(fragmentShader);
        GL.DeleteShader(vertexShader);

        //шейдер готовий до використання
        //тепер оптимізуємо використання закешувавши однорідні розташування шейдерів
        //оскільки кверінг у шейдера дуже повільний ми зробимо це один раз при
ініціалізаціїї
        //і використовуватимо ці значення пізніше

        // спочатку отримаємо кількість активних юніформ у шейдері
        GL.GetProgram(Handle, GetProgramParameterName.ActiveUniforms, out var number-
OfUnifotms);

        //виділимо словник для зберігання розташувань
        _uniformLocations = new Dictionary<string, int>();

        // перебір усіх юніформ шейдерної програми
        for(var i=0; i < numberOfUnifotms; i++)
        {
            // отримуємо ім'я юніформи
            var key = GL.GetActiveUniform(Handle, i, out _, out _);

            //отримуємо її розташування
            var location = GL.GetUniformLocation(Handle, key);

            // і додаємо це все до словника
            _uniformLocations.Add(key, location);
        }
        // ініціалізатор шейдеру готовий
    }

    // напишемо метод компілювання шейдеру з перевіркою на успішність
    private static void CompileShader(int shader)
    {
        // компілюємо шейдер
        GL.CompileShader(shader);

        // перевіряємо на помилку компіляції, оскільки при невдачному компілюванні
функція
        // не викидає помилку тому потрібно перевіряти статус компіляції
        GL.GetShader(shader, ShaderParameter.CompileStatus, out var status);//статус
компіляціїї знаходиться в перелічені параметрів шейдеру
        if (status != (int)All.True)
        {
            // тут перевірка статусу компіляції
            // якщо помилка кидаємо лог помилки для детальної інформації
            // та з яким шейдером проблема
            var infoLog = GL.GetShaderInfoLog(shader);
            throw new Exception($"Error occurred whilst compiling
Shader({shader}).\n\n{infoLog}");
        }
    }

    // напишемо лінковщик програми з перевіркою на успішність лінкування
    private static void LinkProgram(int program)
    {
        // лінкуємо програму
        GL.LinkProgram(program);

        // перевірка на помилки лінкування
        GL.GetProgram(program, GetProgramParameterName.LinkStatus, out var status);
        if (status != (int)All.True)
        {
            // аналогічно до компіляції шейдера
            var infoLog = GL.GetProgramInfoLog(program);
```

```
            throw new Exception($"Error occurred whilst compiling
Shader({program}).\n\n{infoLog}");
        }

    }

    //інкапсулюємо увімкнення шейдерної програми
    public void UseShader()
    {
        GL.UseProgram(Handle);
    }

    // отримаємо розташування
    public int GetAttribLocation(string attribName)
    {
        return GL.GetAttribLocation(Handle, attribName);
    }

    // Сетери юніформ
    //  1.Прив'язуємо програму для юніформи
    //  2.Отримуємо маркер розташування юніформи
    //  3.Використовуємо відповідну функцію GL.Uniform* щоб встановити форму

    /// <summary>
    /// Встановлення int-юніформи для цього шейдера
    /// </summary>
    /// <param name="name">Ім'я юніформи</param>
    /// <param name="data">Данні для встановлення</param>
    public void SetInt(string name, int data)
    {
        GL.UseProgram(Handle);
        GL.Uniform1(_uniformLocations[name], data);
    }

    /// <summary>
    /// Встановлення float-юніформи для цього шейдера
    /// </summary>
    /// <param name="name">Ім'я юніформи</param>
    /// <param name="data">Данні для встановлення</param>
    public void SetFloat(string name, float data)
    {
        GL.UseProgram(Handle);
        GL.Uniform1(_uniformLocations[name], data);
    }

    /// <summary>
    /// Встановлення Matrix4-юніформи для цього шейдера
    /// </summary>
    /// <param name="name">Ім'я юніформи</param>
    /// <param name="data">Данні для встановлення</param>
    /// <remarks>
    ///   <para>
    ///   Матриця транспонується перед надсиланням дот шейдеру
    ///   </para>
    /// </remarks>
    public void SetMatrix4(string name, Matrix4 data)
    {
        GL.UseProgram(Handle);
        GL.UniformMatrix4(_uniformLocations[name], true, ref data);
    }

    /// <summary>
    /// Встановлення Vector3-юніформи для цього шейдера
    /// </summary>
    /// <param name="name">Ім'я юніформи</param>
```

```
    /// <param name="data">Данні для встановлення</param>
    public void SetVector3(string name, Vector3 data)
    {
        GL.UseProgram(Handle);
        GL.Uniform3(_uniformLocations[name], data);
    }
    //клас шейдеру готовий
}
```



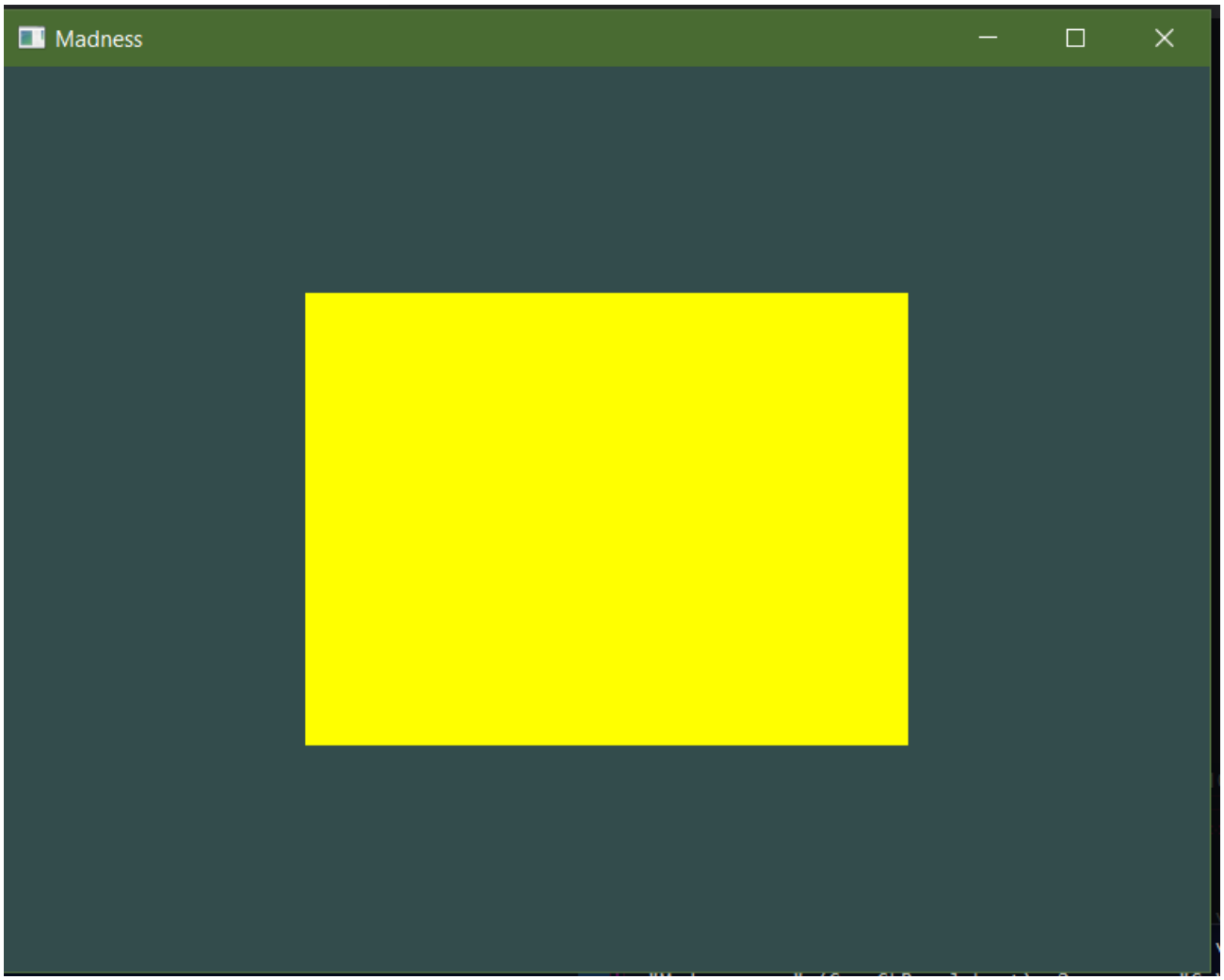
3.Відмалюємо квадрат:

Window.cs

```
// створимо EBO
public class Window : GameWindow
{
    // розширюємо масив вершин
    private readonly float[] _vertices =
    {
             0.5f,  0.5f, 0.0f, // верхній правий
             0.5f, -0.5f, 0.0f, // нижній правий
            -0.5f, -0.5f, 0.0f, // нижній лівий
            -0.5f,  0.5f, 0.0f, // верхній лівий
        };

    // створимо масив індексів
    private readonly uint[] _indices =
    {
            0, 1, 3, // перший трикутник
            1, 2, 3  // другий трикутник
        };

    private int _vertexBufferObject;

    private int _vertexArrayObject;
```

```
    private Shader _shader;

    // створимо вказівник EBO
    private int _elementBufferObject;

    public Window(GameWindowSettings gameWindowSettings, NativeWindowSettings nativeWin-
dowSettings)
        : base(gameWindowSettings, nativeWindowSettings)
    {
    }

    protected override void OnLoad()
    {
        base.OnLoad();

        GL.ClearColor(0.2f, 0.3f, 0.3f, 1.0f);

        _vertexBufferObject = GL.GenBuffer();
        GL.BindBuffer(BufferTarget.ArrayBuffer, _vertexBufferObject);
        GL.BufferData(BufferTarget.ArrayBuffer, _vertices.Length * sizeof(float),
_vertices, BufferUsageHint.StaticDraw);

        _vertexArrayObject = GL.GenVertexArray();
        GL.BindVertexArray(_vertexArrayObject);

        GL.VertexAttribPointer(0, 3, VertexAttribPointerType.Float, false, 3 *
sizeof(float), 0);
        GL.EnableVertexAttribArray(0);

        // генеруємо EBO
        _elementBufferObject = GL.GenBuffer();
        GL.BindBuffer(BufferTarget.ElementArrayBuffer, _elementBufferObject);
        // заносимо данні в EBO
        GL.BufferData(BufferTarget.ElementArrayBuffer, _indices.Length * sizeof(uint),
_indices, BufferUsageHint.StaticDraw);

        _shader = new Shader("Shaders/shader.vert", "Shaders/shader.frag");
        _shader.Use();
    }

    protected override void OnRenderFrame(FrameEventArgs e)
    {
        base.OnRenderFrame(e);

        GL.Clear(ClearBufferMask.ColorBufferBit);

        _shader.Use();

        GL.BindVertexArray(_vertexArrayObject);

        // DrawElements
        // Аргументи:
        // Примітивний тип для малювання. Трикутники в даному випадку.
        // Скільки індексів потрібно намалювати. У цьому випадку шість.
        // Тип даних індексів. Індекси є unsigned int, тому ми також хочемо, щоб це було
тут.
        // Зміщення в EBO. Встановіть значення 0, оскільки ми хочемо намалювати все.
        GL.DrawElements(PrimitiveType.Triangles, _indices.Length, DrawElement-
sType.UnsignedInt, 0);

        SwapBuffers();
    }

    protected override void OnUpdateFrame(FrameEventArgs e)
    {
```

```
        base.OnUpdateFrame(e);

        var input = KeyboardState;

        if (input.IsKeyDown(Keys.Escape))
        {
            Close();
        }
    }

    protected override void OnResize(ResizeEventArgs e)
    {
        base.OnResize(e);
        GL.Viewport(0, 0, Size.X, Size.Y);
    }
}
```



4.Робота з шейдерами

shader.vert

```
#version 330 core

// позиція змінної 0
layout(location = 0) in vec3 aPosition;

// ключове слово out передає значення в наступне по ланцюжку значення
out vec4 vertexColor;

void main(void)
{
```

```
    gl_Position = vec4(aPosition, 1.0);

      // передамо темночервоний колір
      vertexColor = vec4(0.5, 0.0, 0.0, 1.0);
      }
```

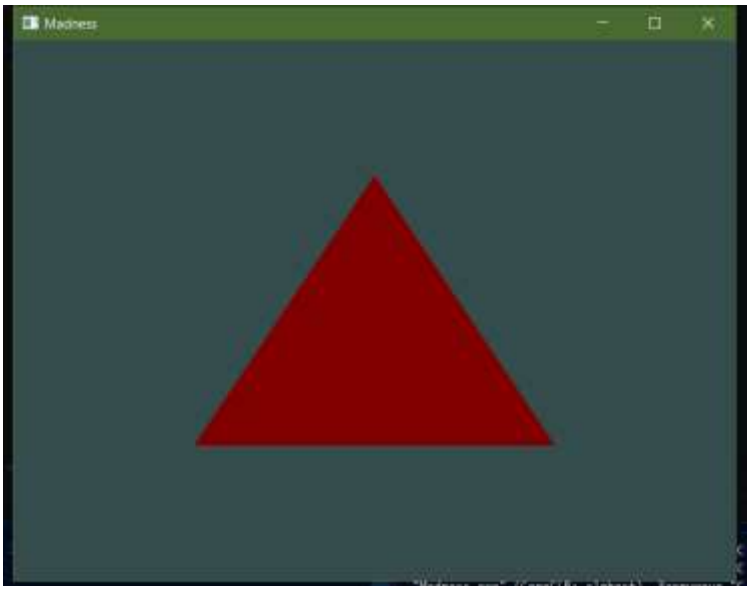
shader.frag

```
#version 330 core

out vec4 outputColor;

// створимо вхідну змінну кольору

in vec4 vertexColor;

void main()
{
    outputColor = vertexColor;
      }
```

Window.cs

```
    // проведемо роботу з шейдерами
    public class Window : GameWindow
    {

        private readonly float[] _vertices =
        {
            -0.5f, -0.5f, 0.0f,
             0.5f, -0.5f, 0.0f,
             0.0f,  0.5f, 0.0f
        };

        private int _vertexBufferObject;

        private int _vertexArrayObject;

        private Shader _shader;

        public Window(GameWindowSettings gameWindowSettings, NativeWindowSettings native-
WindowSettings)
            : base(gameWindowSettings, nativeWindowSettings)
        {
        }

        protected override void OnLoad()
        {
            base.OnLoad();

            GL.ClearColor(0.2f, 0.3f, 0.3f, 1.0f);

            _vertexBufferObject = GL.GenBuffer();

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

            _vertexArrayObject = GL.GenVertexArray();
            GL.BindVertexArray(_vertexArrayObject);

            GL.VertexAttribPointer(0, 3, VertexAttribPointerType.Float, false, 3 *
sizeof(float), 0);
            GL.EnableVertexAttribArray(0);

            // тут ми перевіряємо, скільки атрибутів вершин може обробляти наше обладнан-
ня.
            GL.GetInteger(GetPName.MaxVertexAttribs, out int maxAttributeCount);
```

```
            Debug.WriteLine($"Maximum number of vertex attributes supported: {maxAt-
tributeCount}");

            _shader = new Shader("Shaders/shader.vert", "Shaders/shader.frag");
            _shader.Use();
        }

        protected override void OnRenderFrame(FrameEventArgs e)
        {
            base.OnRenderFrame(e);

            GL.Clear(ClearBufferMask.ColorBufferBit);

            _shader.Use();

            GL.BindVertexArray(_vertexArrayObject);

            GL.DrawArrays(PrimitiveType.Triangles, 0, 3);

            SwapBuffers();
        }

        protected override void OnUpdateFrame(FrameEventArgs e)
        {
            base.OnUpdateFrame(e);

            var input = KeyboardState;

            if (input.IsKeyDown(Keys.Escape))
            {
                Close();
            }
        }

        protected override void OnResize(ResizeEventArgs e)
        {
            base.OnResize(e);

            GL.Viewport(0, 0, Size.X, Size.Y);
        }
    }
```



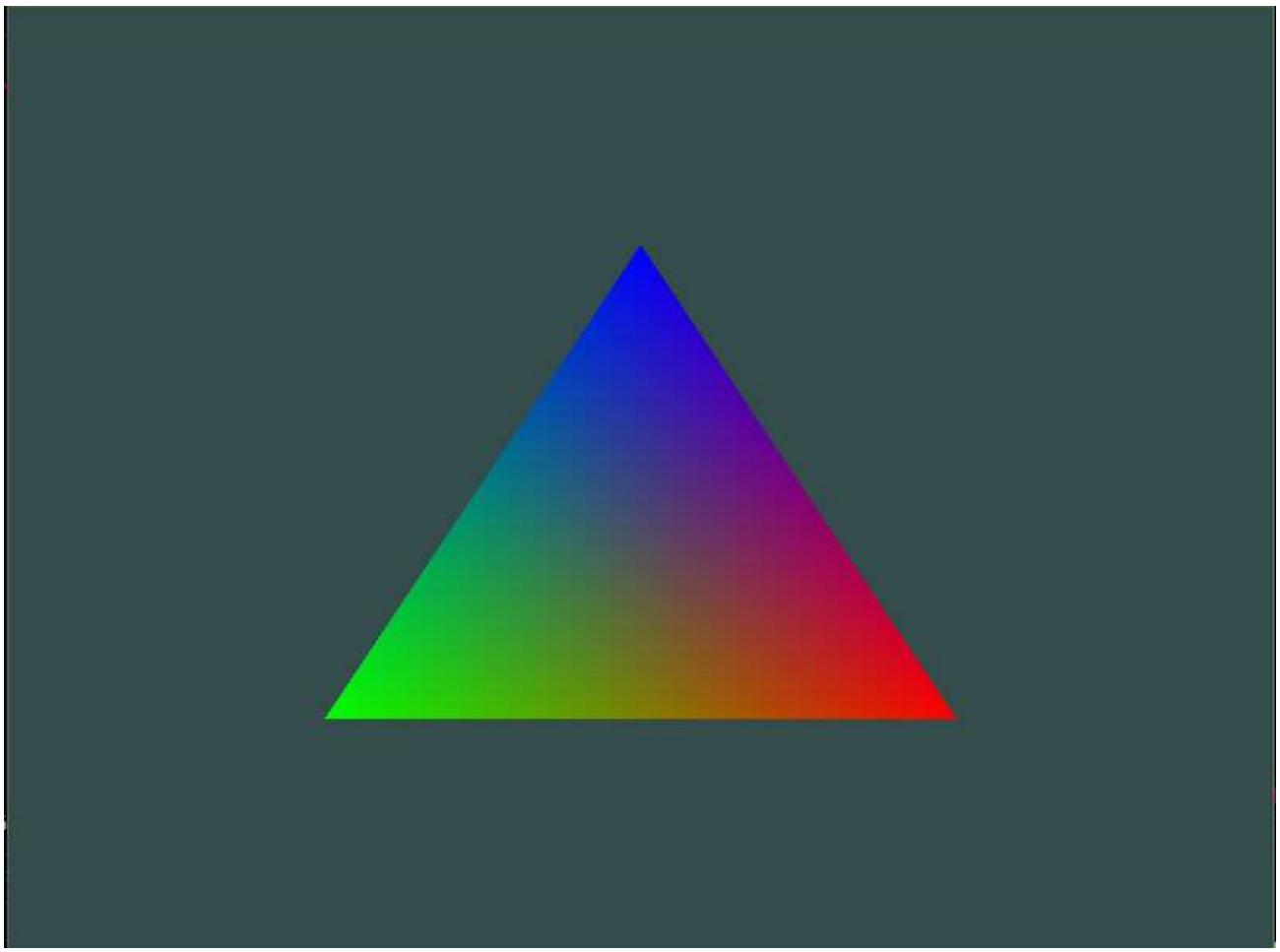
shader.vert

```
#version 330 core

// атрибут позиції 0
layout(location = 0) in vec3 aPosition;

// сюди потрапляють значення кольорів які призначено в основній програмі
layout(location = 1) in vec3 aColor;

out vec3 ourColor; // передаємо колір до фрагментного шейдеру

void main(void)
{
    gl_Position = vec4(aPosition, 1.0);

      // використаємо змінну outColor щоб передати інформацію про колір фрагментарному
шейдеру
      ourColor = aColor;
      }
```

Window.cs

```
public class Window : GameWindow
{

    // тут попрацюємо з кольорами
    private readonly float[] _vertices =
    {
             // позиція           // кольор
             0.5f, -0.5f, 0.0f,  1.0f, 0.0f, 0.0f,   // нижній правий
            -0.5f, -0.5f, 0.0f,  0.0f, 1.0f, 0.0f,   // нижній лівий
             0.0f,  0.5f, 0.0f,  0.0f, 0.0f, 1.0f    // верхній
        };

    private int _vertexBufferObject;

    private int _vertexArrayObject;

    private Shader _shader;

    public Window(GameWindowSettings gameWindowSettings, NativeWindowSettings nativeWin-
dowSettings)
        : base(gameWindowSettings, nativeWindowSettings)
    {
    }

    // проініціалізуємо OpenGL
    protected override void OnLoad()
    {
        base.OnLoad();

        GL.ClearColor(0.2f, 0.3f, 0.3f, 1.0f);

        _vertexBufferObject = GL.GenBuffer();

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

        _vertexArrayObject = GL.GenVertexArray();
        GL.BindVertexArray(_vertexArrayObject);

        // створюємо вказівник для 3 позиційних компонентів наших вершин.
        // єдина відмінність тут полягає в тому, що нам потрібно врахувати значення 3
кольорів у змінній stride
```

```
        GL.VertexAttribPointer(0, 3, VertexAttribPointerType.Float, false, 6 *
sizeof(float), 0);
        GL.EnableVertexAttribArray(0);

        // створюємо новий покажчик для значень кольорів.
        // подібно до попереднього покажчика, ми призначаємо 6 у значенні кроку.
        // нам також потрібно правильно встановити зсув, щоб отримати значення кольорів.
        // дані кольору починаються після даних позиції, тому зміщення дорівнює 3 фло-
атам.
        GL.VertexAttribPointer(1, 3, VertexAttribPointerType.Float, false, 6 *
sizeof(float), 3 * sizeof(float));
        // Потім ми вмикаємо атрибут кольору (location=1), щоб він був доступний для шей-
дера.
        GL.EnableVertexAttribArray(1);

        GL.GetInteger(GetPName.MaxVertexAttribs, out int maxAttributeCount);
        Debug.WriteLine($"Maximum number of vertex attributes supported: {maxAt-
tributeCount}");

        _shader = new Shader("Shaders/shader.vert", "Shaders/shader.frag");
        _shader.Use();
    }

    protected override void OnRenderFrame(FrameEventArgs e)
    {
        base.OnRenderFrame(e);

        GL.Clear(ClearBufferMask.ColorBufferBit);

        _shader.Use();

        GL.BindVertexArray(_vertexArrayObject);

        GL.DrawArrays(PrimitiveType.Triangles, 0, 3);

        SwapBuffers();
    }

    protected override void OnUpdateFrame(FrameEventArgs e)
    {
        base.OnUpdateFrame(e);

        var input = KeyboardState;

        if (input.IsKeyDown(Keys.Escape))
        {
            Close();
        }
    }

    protected override void OnResize(ResizeEventArgs e)
    {
        base.OnResize(e);

        GL.Viewport(0, 0, Size.X, Size.Y);
    }
}
```



Window.cs

```
public class Window : GameWindow
{
    // тут розглядено уніфіковані типи змінних
    private readonly float[] _vertices =
    {
            -0.5f, -0.5f, 0.0f,
             0.5f, -0.5f, 0.0f,
             0.0f,  0.5f, 0.0f
        };

    // змусими трикутник пульсувати міє певним діапазоном кольорів
    // для цього нам потрібен таймер адже вінпостійно зростає
    private Stopwatch _timer;

    private int _vertexBufferObject;

    private int _vertexArrayObject;

    private Shader _shader;

    public Window(GameWindowSettings gameWindowSettings, NativeWindowSettings nativeWin-
dowSettings)
        : base(gameWindowSettings, nativeWindowSettings)
    {
    }

    protected override void OnLoad()
    {
        base.OnLoad();

        GL.ClearColor(0.2f, 0.3f, 0.3f, 1.0f);
```

```
        _vertexBufferObject = GL.GenBuffer();

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

        _vertexArrayObject = GL.GenVertexArray();
        GL.BindVertexArray(_vertexArrayObject);

        GL.VertexAttribPointer(0, 3, VertexAttribPointerType.Float, false, 3 *
sizeof(float), 0);
        GL.EnableVertexAttribArray(0);

        GL.GetInteger(GetPName.MaxVertexAttribs, out int maxAttributeCount);
        Debug.WriteLine($"Maximum number of vertex attributes supported: {maxAt-
tributeCount}");

        _shader = new Shader("Shaders/shader.vert", "Shaders/shader.frag");
        _shader.Use();

        // запускаємо секундомір
        _timer = new Stopwatch();
        _timer.Start();
    }

    protected override void OnRenderFrame(FrameEventArgs e)
    {
        base.OnRenderFrame(e);

        GL.Clear(ClearBufferMask.ColorBufferBit);

        _shader.Use();

        // тут ми отримуємо загальну кількість секунд, що минули з моменту останнього
скидання цього методу
        // і ми призначаємо його змінній timeValue, щоб його можна було використовувати
для пульсуючого кольору
        double timeValue = _timer.Elapsed.TotalSeconds;

        // оскільки синус набуває значень між -1 та 1 зробимо синусоїдальну зміну кольору
        float greenValue = (float)Math.Sin(timeValue) / 2.0f + 0.5f;

        // отримуємо уніфіковану змінну з фрагментного шейдеру
        int vertexColorLocation = GL.GetUniformLocation(_shader.Handle, "ourColor");

        // тут ми призначаємо змінну ourColor у фрагментному шейдері
        // через метод OpenGL Uniform, який приймає значення як окремі значення vec
        GL.Uniform4(vertexColorLocation, 0.0f, greenValue, 0.0f, 1.0f);
        // GL.Uniform4(vertexColorLocation, new OpenTK.Mathematics.Color4(0f, greenValue,
0f, 0f));

        GL.BindVertexArray(_vertexArrayObject);

        GL.DrawArrays(PrimitiveType.Triangles, 0, 3);

        SwapBuffers();
    }

    protected override void OnUpdateFrame(FrameEventArgs e)
    {
        base.OnUpdateFrame(e);

        var input = KeyboardState;
```

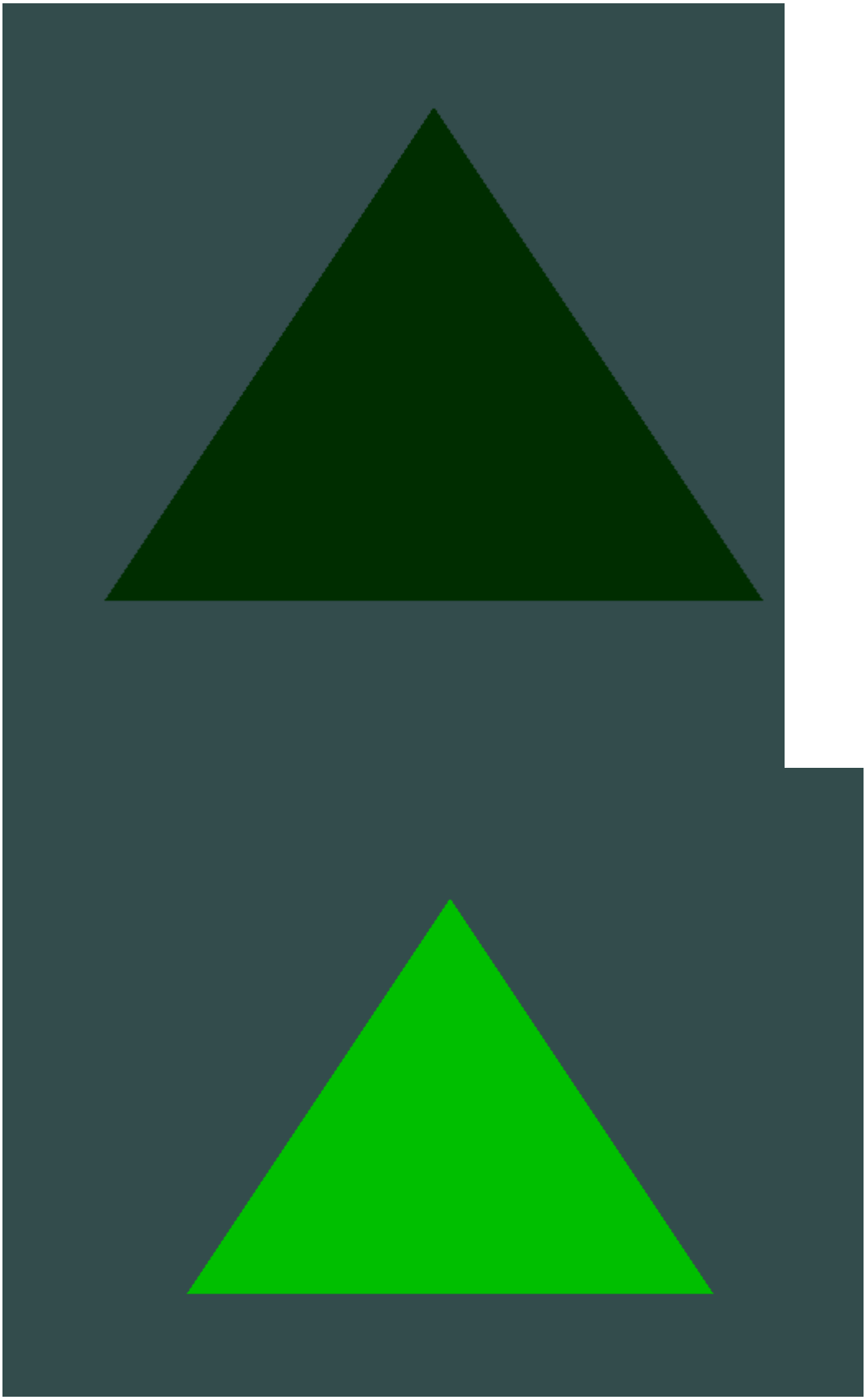
```
        if (input.IsKeyDown(Keys.Escape))
        {
            Close();
        }
    }

    protected override void OnResize(ResizeEventArgs e)
    {
        base.OnResize(e);

        GL.Viewport(0, 0, Size.X, Size.Y);
    }
}
```

shader.frag

```
#version 330 core

out vec4 outputColor;

// ключове слово Uniform дозволяє отримати доступ до змінної шейдера на будь-якому етапі ланцюжка шейдерів
uniform vec4 ourColor;

void main()
{
    outputColor = ourColor;
      }
```

shader.vert

```
#version 330 core

layout(location = 0) in vec3 aPosition;

void main(void)
{
    gl_Position = vec4(aPosition, 1.0);
      }
```



****** пульсація кольору від чорного до зеленого ******

5. текстури

Window.cs

```
// Оскільки ми додаємо текстуру, ми змінюємо масив вершин, щоб включити координати тек-
стури.
    // Координати текстури в діапазоні від 0,0 до 1,0, де (0,0, 0,0) представляють нижній
лівий кут, а (1,0, 1,0) представляють верхній правий.
    private readonly float[] _vertices =
    {
            // Координати               Координати текстури
             0.5f,  0.5f, 0.0f,         1.0f, 1.0f,
             0.5f, -0.5f, 0.0f,         1.0f, 0.0f,
            -0.5f, -0.5f, 0.0f,         0.0f, 0.0f,
            -0.5f,  0.5f, 0.0f,         0.0f, 1.0f
        };

    private readonly uint[] _indices =
    {
            0, 1, 3,
            1, 2, 3
        };

    private int _elementBufferObject;

    private int _vertexBufferObject;

    private int _vertexArrayObject;

    private Shader _shader;

    private Texture _texture;

    public Window(GameWindowSettings gameWindowSettings, NativeWindowSettings nativeWin-
dowSettings)
        : base(gameWindowSettings, nativeWindowSettings)
    {
    }

    protected override void OnLoad()
    {
        base.OnLoad();

        GL.ClearColor(0.2f, 0.3f, 0.3f, 1.0f);

        _vertexArrayObject = GL.GenVertexArray();
        GL.BindVertexArray(_vertexArrayObject);

        _vertexBufferObject = GL.GenBuffer();
        GL.BindBuffer(BufferTarget.ArrayBuffer, _vertexBufferObject);
        GL.BufferData(BufferTarget.ArrayBuffer, _vertices.Length * sizeof(float),
_vertices, BufferUsageHint.StaticDraw);

        _elementBufferObject = GL.GenBuffer();
        GL.BindBuffer(BufferTarget.ElementArrayBuffer, _elementBufferObject);
        GL.BufferData(BufferTarget.ElementArrayBuffer, _indices.Length * sizeof(uint),
_indices, BufferUsageHint.StaticDraw);

        _shader = new Shader("Shaders/shader.vert", "Shaders/fragshader.frag");
        _shader.Use();

        // Оскільки між початком першої вершини та початком другої тепер є 5 флоатів,
        // ми змінюємо крок від 3 * sizeof(float) до 5 * sizeof(float).
        // Це тепер передасть новий масив вершин до буфера.
        var vertexLocation = _shader.GetAttribLocation("aPosition");
        GL.EnableVertexAttribArray(vertexLocation);
```

```
        GL.VertexAttribPointer(vertexLocation, 3, VertexAttribPointerType.Float, false, 5
* sizeof(float), 0);

        // Далі ми також встановлюємо координати текстури. Це працює приблизно так само.
        // Ми додаємо зсув 3, оскільки координати текстури йдуть після даних позиції.
        // Ми також змінюємо кількість даних на 2, тому що для координат текстури є лише
2 числа з плаваючою точкою.
        var texCoordLocation = _shader.GetAttribLocation("aTexCoord");
        GL.EnableVertexAttribArray(texCoordLocation);
        GL.VertexAttribPointer(texCoordLocation, 2, VertexAttribPointerType.Float, false,
5 * sizeof(float), 3 * sizeof(float));

        _texture = Texture.LoadFromFile("Resources/container.png");
        _texture.Use(TextureUnit.Texture0);
    }

    protected override void OnRenderFrame(FrameEventArgs e)
    {
        base.OnRenderFrame(e);

        GL.Clear(ClearBufferMask.ColorBufferBit);

        GL.BindVertexArray(_vertexArrayObject);

        _texture.Use(TextureUnit.Texture0);
        _shader.Use();

        GL.DrawElements(PrimitiveType.Triangles, _indices.Length, DrawElement-
sType.UnsignedInt, 0);

        SwapBuffers();
    }

    protected override void OnUpdateFrame(FrameEventArgs e)
    {
        base.OnUpdateFrame(e);

        var input = KeyboardState;

        if (input.IsKeyDown(Keys.Escape))
        {
            Close();
        }
    }

    protected override void OnResize(ResizeEventArgs e)
    {
        base.OnResize(e);

        GL.Viewport(0, 0, Size.X, Size.Y);
    }
}
```

shader.frag

```
#version 330

out vec4 outputColor;

in vec2 texCoord;

uniform sampler2D texture0;

void main()
{
```

```
    outputColor = texture(texture0, texCoord);
    }
```

shader.vert

```
#version 330 core

layout(location = 0) in vec3 aPosition;

// Ми додаємо ще одну вхідну змінну для координат текстури.
layout(location = 1) in vec2 aTexCoord;

// Однак вони не потрібні для самого вершинного шейдера.
// Замість цього ми створюємо вихідну змінну, щоб ми могли надіслати ці дані до фрагмент-
ного шейдера.

out vec2 texCoord;

void main(void)
{
    // Потім ми додаємо вхідну текстурну координату до вихідної.
    // texCoord тепер можна використовувати у фрагментному шейдері.
    texCoord = aTexCoord;

    gl_Position = vec4(aPosition, 1.0);
    }
```

Texture.cs

```
public class Texture
{
    public readonly int Handle;

    public static Texture LoadFromFile(string path)
    {
        // згенеруємо вказівник
        int handle = GL.GenTexture();

        // прив'яжемо його
        GL.ActiveTexture(TextureUnit.Texture0);
        GL.BindTexture(TextureTarget.Texture2D, handle);

        // OpenGL має своє транслювання текстури в нижньому лівому куті замість верхнього
лівого кута,
        // тому ми використаємо StbImageSharp щоб перевертати зображення під час заванта-
ження.
        StbImage.stbi_set_flip_vertically_on_load(1);

        // Тут ми відкриваємо потік до файлу та передаємо його для завантаження StbImage-
Sharp.
        using (Stream stream = File.OpenRead(path))
        {
            ImageResult image = ImageResult.FromStream(stream, ColorCompo-
nents.RedGreenBlueAlpha);

            // тепер створимо текстуру
            // GL.TexImage2D.
            // Аргументи:
            // Тип текстури, яку ми створюємо. Існує багато різних типів текстур, але
єдина, яка нам зараз потрібна, це Texture2D.
            // Рівень деталізації.
            // Цільовий формат пікселів. Це формат, у якому OpenGL зберігатиме зображен-
ня.
            // Ширина зображення
            // Висота зображення.
            // Границя зображення. Це завжди має бути 0; це застарілий параметр
            // Формат пікселів
            // Тип даних пікселів.
```

```
            // пікселі.
            GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba,
image.Width, image.Height, 0, PixelFormat.Rgba, PixelType.UnsignedByte, image.Data);
        }

        // спочатку ми встановлюємо min і mag фільтр. Вони використовуються, коли тексту-
ру зменшено та збільшено відповідно.
        GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter,
(int)TextureMinFilter.Linear);
        GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter,
(int)TextureMagFilter.Linear);

        // встановимо режим обтікання. S для осі X, а T для осі Y.
        // встановимо значення Repeat, щоб текстури повторювалися під час обертання
        GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapS,
(int)TextureWrapMode.Repeat);
        GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapT,
(int)TextureWrapMode.Repeat);

        // Далі генеруємо mipmaps.
        // Mipmaps — це зменшені копії текстури. Кожен рівень mipmap вдвічі менший за
попередній
        // запобігає ефекту морье
        GL.GenerateMipmap(GenerateMipmapTarget.Texture2D);

        return new Texture(handle);
    }

    public Texture(int glHandle)
    {
        Handle = glHandle;
    }

    // Активація текстури
    // використаємо GL.ActiveTexture, щоб установити, до якого слота прив'язується
GL.BindTexture.
    public void Use(TextureUnit unit)
    {
        GL.ActiveTexture(unit);
        GL.BindTexture(TextureTarget.Texture2D, Handle);
    }
   }
```

Window.cs

```
public class Window : GameWindow
{
    // Оскільки ми додаємо текстуру, ми змінюємо масив вершин, щоб включити координати
текстури.
    // Координати текстури в діапазоні від 0,0 до 1,0, де (0,0, 0,0) представляють нижній
лівий кут, а (1,0, 1,0) представляють верхній правий.
    private readonly float[] _vertices =
    {
            // Координати               Координати текстури
             0.5f,  0.5f, 0.0f,         1.0f, 1.0f,
             0.5f, -0.5f, 0.0f,         1.0f, 0.0f,
            -0.5f, -0.5f, 0.0f,         0.0f, 0.0f,
            -0.5f,  0.5f, 0.0f,         0.0f, 1.0f
        };

    private readonly uint[] _indices =
    {
            0, 1, 3,
            1, 2, 3
        };

    private int _elementBufferObject;

    private int _vertexBufferObject;

    private int _vertexArrayObject;

    private Shader _shader;

    private Texture _texture;
    private Texture _texture2;

    public Window(GameWindowSettings gameWindowSettings, NativeWindowSettings nativeWin-
dowSettings)
        : base(gameWindowSettings, nativeWindowSettings)
    {
    }

    protected override void OnLoad()
    {
        base.OnLoad();

        GL.ClearColor(0.2f, 0.3f, 0.3f, 1.0f);

        _vertexArrayObject = GL.GenVertexArray();
        GL.BindVertexArray(_vertexArrayObject);

        _vertexBufferObject = GL.GenBuffer();
        GL.BindBuffer(BufferTarget.ArrayBuffer, _vertexBufferObject);
        GL.BufferData(BufferTarget.ArrayBuffer, _vertices.Length * sizeof(float),
_vertices, BufferUsageHint.StaticDraw);

        _elementBufferObject = GL.GenBuffer();
        GL.BindBuffer(BufferTarget.ElementArrayBuffer, _elementBufferObject);
        GL.BufferData(BufferTarget.ElementArrayBuffer, _indices.Length * sizeof(uint),
_indices, BufferUsageHint.StaticDraw);

        // щось через коментарі шейдери перестали працювати
        // тому коментуватиму зміни тут
        // додано вхідний параметер для другої текстури
        _shader = new Shader("Shaders/shader.vert", "Shaders/fragshader.frag");
        _shader.Use();
```

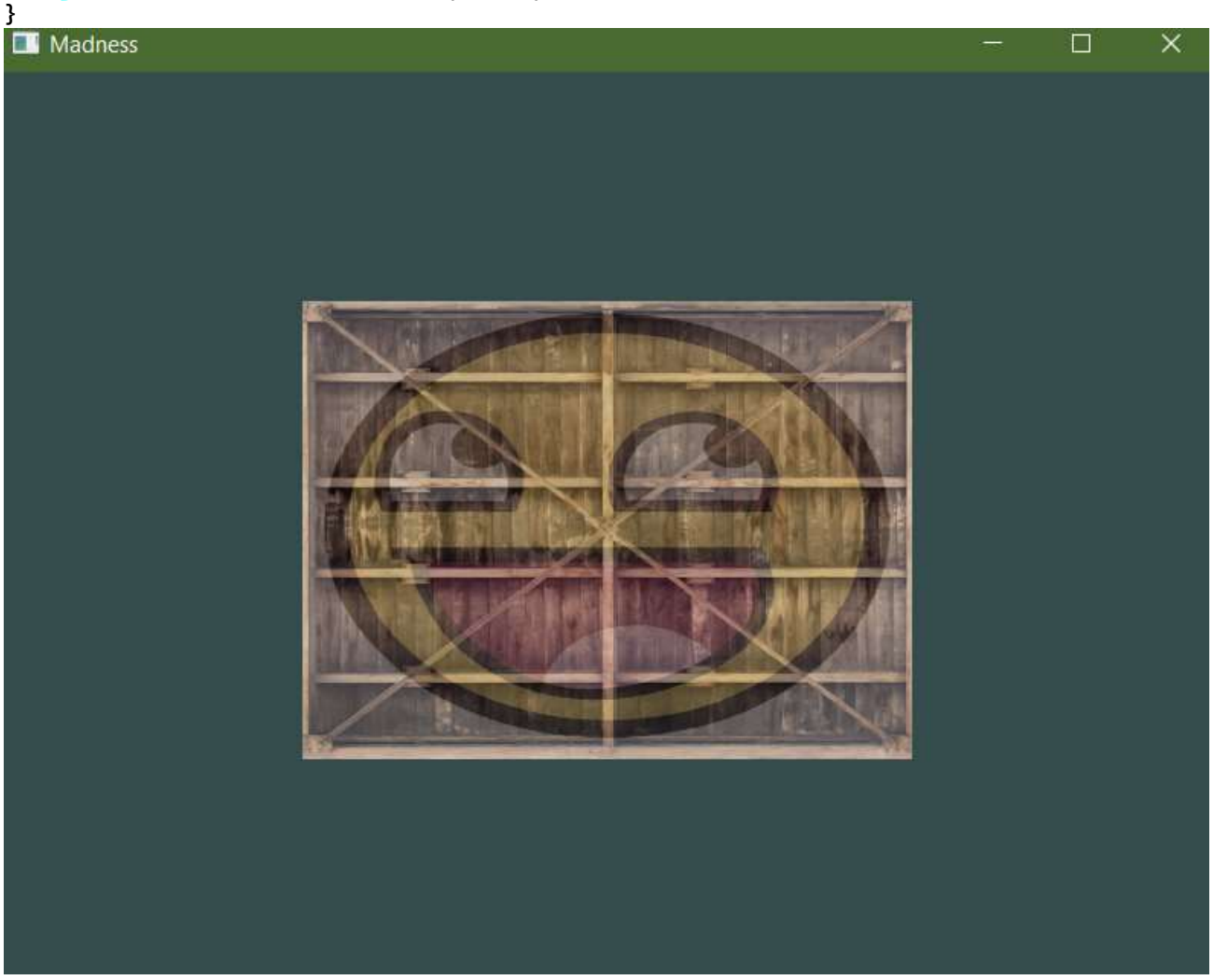
```
        var vertexLocation = _shader.GetAttribLocation("aPosition");
        GL.EnableVertexAttribArray(vertexLocation);
        GL.VertexAttribPointer(vertexLocation, 3, VertexAttribPointerType.Float, false, 5
* sizeof(float), 0);

        var texCoordLocation = _shader.GetAttribLocation("aTexCoord");
        GL.EnableVertexAttribArray(texCoordLocation);
        GL.VertexAttribPointer(texCoordLocation, 2, VertexAttribPointerType.Float, false,
5 * sizeof(float), 3 * sizeof(float));

        _texture = Texture.LoadFromFile("Resources/container.png");
        // Перша текстура йде в блок текстури 0.
        _texture.Use(TextureUnit.Texture0);

        // оскільки System.Drawing читає пікселі не так як очікує OpenGL
        _texture2 = Texture.LoadFromFile("Resources/awesomeface.png");
        // друга текстура йде в блок текстури 1.
        _texture2.Use(TextureUnit.Texture1);

        // Далі ми повинні налаштувати вибірки в шейдерах для використання правильних
текстур.
        // Int, який ми надсилаємо у uniform, вказує, яку текстурну одиницю має викори-
стовувати семплер.
        _shader.SetInt("texture0", 0);
        _shader.SetInt("texture1", 1);
    }

    protected override void OnRenderFrame(FrameEventArgs e)
    {
        base.OnRenderFrame(e);

        GL.Clear(ClearBufferMask.ColorBufferBit);

        GL.BindVertexArray(_vertexArrayObject);

        _texture.Use(TextureUnit.Texture0);
        _shader.Use();

        GL.DrawElements(PrimitiveType.Triangles, _indices.Length, DrawElement-
sType.UnsignedInt, 0);

        SwapBuffers();
    }

    protected override void OnUpdateFrame(FrameEventArgs e)
    {
        base.OnUpdateFrame(e);

        var input = KeyboardState;

        if (input.IsKeyDown(Keys.Escape))
        {
            Close();
        }
    }

    protected override void OnResize(ResizeEventArgs e)
    {
        base.OnResize(e);

        GL.Viewport(0, 0, Size.X, Size.Y);
    }
}
```

shader.frag

```
#version 330

out vec4 outputColor;

in vec2 texCoord;

uniform sampler2D texture0;
uniform sampler2D texture1;

void main()
{
    outputColor = mix(texture(texture0, texCoord), texture(texture1, texCoord), 0.2);
      }
```

shader.vert

```
#version 330 core

layout(location = 0) in vec3 aPosition;
layout(location = 1) in vec2 aTexCoord;

out vec2 texCoord;

void main(void)
{
    texCoord = aTexCoord;

    gl_Position = vec4(aPosition, 1.0);
}
```



6. Робота з трансформаціями

Window.cs

```
public class Window : GameWindow
{
    // Оскільки ми додаємо текстуру, ми змінюємо масив вершин, щоб включити координати
текстури.
    // Координати текстури в діапазоні від 0,0 до 1,0, де (0,0, 0,0) представляють нижній
лівий кут, а (1,0, 1,0) представляють верхній правий.
    private readonly float[] _vertices =
    {
            // Координати               Координати текстури
             0.5f,  0.5f, 0.0f,         1.0f, 1.0f,
             0.5f, -0.5f, 0.0f,         1.0f, 0.0f,
            -0.5f, -0.5f, 0.0f,         0.0f, 0.0f,
            -0.5f,  0.5f, 0.0f,         0.0f, 1.0f
        };

    private readonly uint[] _indices =
    {
            0, 1, 3,
            1, 2, 3
        };

    private int _elementBufferObject;

    private int _vertexBufferObject;

    private int _vertexArrayObject;

    private Shader _shader;

    private Texture _texture;
    private Texture _texture2;

    public Window(GameWindowSettings gameWindowSettings, NativeWindowSettings nativeWin-
dowSettings)
        : base(gameWindowSettings, nativeWindowSettings)
    {
    }

    protected override void OnLoad()
    {
        base.OnLoad();

        GL.ClearColor(0.2f, 0.3f, 0.3f, 1.0f);

        _vertexArrayObject = GL.GenVertexArray();
        GL.BindVertexArray(_vertexArrayObject);

        _vertexBufferObject = GL.GenBuffer();
        GL.BindBuffer(BufferTarget.ArrayBuffer, _vertexBufferObject);
        GL.BufferData(BufferTarget.ArrayBuffer, _vertices.Length * sizeof(float),
_vertices, BufferUsageHint.StaticDraw);

        _elementBufferObject = GL.GenBuffer();
        GL.BindBuffer(BufferTarget.ElementArrayBuffer, _elementBufferObject);
        GL.BufferData(BufferTarget.ElementArrayBuffer, _indices.Length * sizeof(uint),
_indices, BufferUsageHint.StaticDraw);

        //у вертексний шейдер додано пераметер трансформації матриці
        _shader = new Shader("Shaders/shader.vert", "Shaders/fragshader.frag");
        _shader.Use();

        var vertexLocation = _shader.GetAttribLocation("aPosition");
```

```
        GL.EnableVertexAttribArray(vertexLocation);
        GL.VertexAttribPointer(vertexLocation, 3, VertexAttribPointerType.Float, false, 5
* sizeof(float), 0);

        var texCoordLocation = _shader.GetAttribLocation("aTexCoord");
        GL.EnableVertexAttribArray(texCoordLocation);
        GL.VertexAttribPointer(texCoordLocation, 2, VertexAttribPointerType.Float, false,
5 * sizeof(float), 3 * sizeof(float));

        _texture = Texture.LoadFromFile("Resources/container.png");

        _texture.Use(TextureUnit.Texture0);


        _texture2 = Texture.LoadFromFile("Resources/awesomeface.png");
        _texture2.Use(TextureUnit.Texture1);

        _shader.SetInt("texture0", 0);
        _shader.SetInt("texture1", 1);
    }

    protected override void OnRenderFrame(FrameEventArgs e)
    {
        base.OnRenderFrame(e);

        GL.Clear(ClearBufferMask.ColorBufferBit);

        GL.BindVertexArray(_vertexArrayObject);

        // створимо одиничну матрицю
        var transform = Matrix4.Identity;

        // Тут ми поєднуємо матрицю перетворення з іншою, створеною OpenTK, щоб повернути
її на 20 градусів.
        transform = transform * Ma-
trix4.CreateRotationZ(MathHelper.DegreesToRadians(20f));

        // Далі ми масштабуємо матрицю.
        transform = transform * Matrix4.CreateScale(1.1f);

        // Потім ми транслюємо матрицю, що трохи переміщує її у верхній правий кут.
        transform = transform * Matrix4.CreateTranslation(0.1f, 0.1f, 0.0f);

        _texture.Use(TextureUnit.Texture0);
        _texture2.Use(TextureUnit.Texture1);
        _shader.Use();

        // уніфікуємо матрицю в шейдер
        _shader.SetMatrix4("transform", transform);

        GL.DrawElements(PrimitiveType.Triangles, _indices.Length, DrawElement-
sType.UnsignedInt, 0);

        SwapBuffers();
    }

    protected override void OnUpdateFrame(FrameEventArgs e)
    {
        base.OnUpdateFrame(e);

        var input = KeyboardState;

        if (input.IsKeyDown(Keys.Escape))
        {
            Close();
```

```
        }
    }

    protected override void OnResize(ResizeEventArgs e)
    {
        base.OnResize(e);

        GL.Viewport(0, 0, Size.X, Size.Y);
    }
}
```

shader.vert

```
#version 330 core

layout(location = 0) in vec3 aPosition;

layout(location = 1) in vec2 aTexCoord;

out vec2 texCoord;

uniform mat4 transform;

void main(void)
{
    texCoord = aTexCoord;

    gl_Position = vec4(aPosition, 1.0) * transform;
}
```

6.Робота з анімаціями підготовка до додавання камери

Window.cs

```
// змусимо квадрат обертатись
public class Window : GameWindow
```

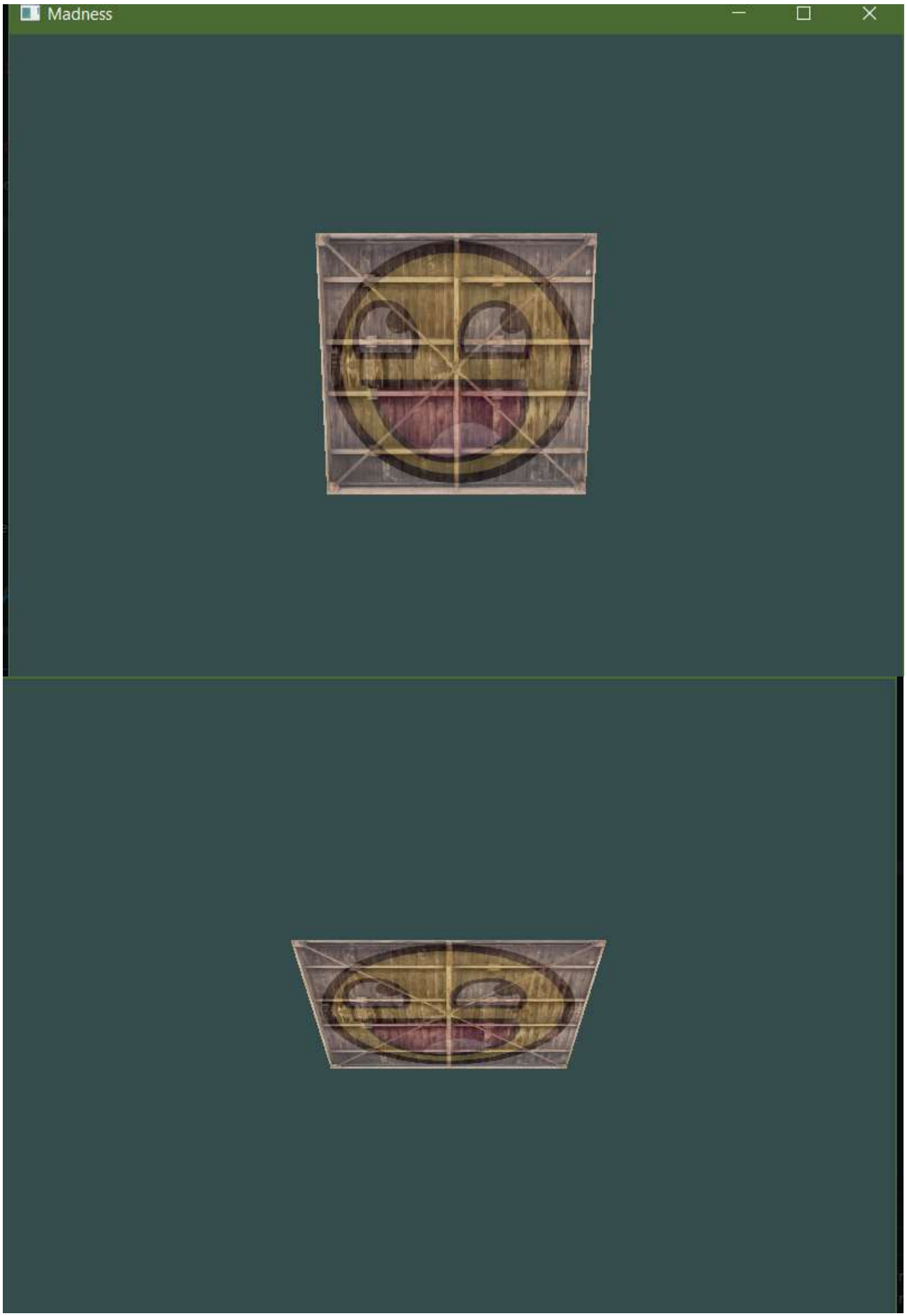
```
{
    private readonly float[] _vertices =
    {
            // позиція              теккстурні координати
             0.5f,  0.5f, 0.0f,      1.0f, 1.0f,
             0.5f, -0.5f, 0.0f,      1.0f, 0.0f,
            -0.5f, -0.5f, 0.0f,      0.0f, 0.0f,
            -0.5f,  0.5f, 0.0f,      0.0f, 1.0f
        };

    private readonly uint[] _indices =
    {
            0, 1, 3,
            1, 2, 3
        };

    private int _elementBufferObject;

    private int _vertexBufferObject;

    private int _vertexArrayObject;

    private Shader _shader;

    private Texture _texture;

    private Texture _texture2;

    // створимо змінну часу, щоб указати, скільки часу минуло з моменту відкриття програ-
ми.
    private double _time;

    // мотім створимо дві матриці для нашого перегляду та проекції
    private Matrix4 _view;
    // це показує, як будуть спроектовані вершини.
    private Matrix4 _projection;

    public Window(GameWindowSettings gameWindowSettings, NativeWindowSettings nativeWin-
dowSettings)
        : base(gameWindowSettings, nativeWindowSettings)
    {
    }

    protected override void OnLoad()
    {
        base.OnLoad();

        GL.ClearColor(0.2f, 0.3f, 0.3f, 1.0f);

        // Тут ми вмикаємо тестування глибини.Ми також очищаємо буфер глибини в GL.Clear
в OnRenderFrame.
        GL.Enable(EnableCap.DepthTest);

        _vertexArrayObject = GL.GenVertexArray();
        GL.BindVertexArray(_vertexArrayObject);

        _vertexBufferObject = GL.GenBuffer();
        GL.BindBuffer(BufferTarget.ArrayBuffer, _vertexBufferObject);
        GL.BufferData(BufferTarget.ArrayBuffer, _vertices.Length * sizeof(float),
_vertices, BufferUsageHint.StaticDraw);

        _elementBufferObject = GL.GenBuffer();
        GL.BindBuffer(BufferTarget.ElementArrayBuffer, _elementBufferObject);
        GL.BufferData(BufferTarget.ElementArrayBuffer, _indices.Length * sizeof(uint),
_indices, BufferUsageHint.StaticDraw);
```

```
        // додано юніфікованні в mat4 змінні model, view , projection
        _shader = new Shader("Shaders/shader.vert", "Shaders/fragshader.frag");
        _shader.Use();

        var vertexLocation = _shader.GetAttribLocation("aPosition");
        GL.EnableVertexAttribArray(vertexLocation);
        GL.VertexAttribPointer(vertexLocation, 3, VertexAttribPointerType.Float, false, 5
* sizeof(float), 0);

        var texCoordLocation = _shader.GetAttribLocation("aTexCoord");
        GL.EnableVertexAttribArray(texCoordLocation);
        GL.VertexAttribPointer(texCoordLocation, 2, VertexAttribPointerType.Float, false,
5 * sizeof(float), 3 * sizeof(float));

        _texture = Texture.LoadFromFile("Resources/container.png");
        _texture.Use(TextureUnit.Texture0);

        _texture2 = Texture.LoadFromFile("Resources/awesomeface.png");
        _texture2.Use(TextureUnit.Texture1);

        _shader.SetInt("texture0", 0);
        _shader.SetInt("texture1", 1);

        //  перемістимо вид на три одиниці назад по осі Z.
        _view = Matrix4.CreateTranslation(0.0f, 0.0f, -3.0f);

        // Для матриці ми використовуємо кілька параметрів.
        //   Поле зору.
        //   Співвідношення сторін.
        //   відсікання ближніх вершин.
        //   відсікання дальніх вершин.
        _projection = Ma-
trix4.CreatePerspectiveFieldOfView(MathHelper.DegreesToRadians(45f), Size.X /
(float)Size.Y, 0.1f, 100.0f);
    }

    protected override void OnRenderFrame(FrameEventArgs e)
    {
        base.OnRenderFrame(e);

        // додаємо час, що минув з останнього кадру, помножений на 4,0 для прискорення
анімації, до загальної кількості часу.
        _time += 4.0 * e.Time;

        // очищаємо буфер глибини
        GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

        GL.BindVertexArray(_vertexArrayObject);

        _texture.Use(TextureUnit.Texture0);
        _texture2.Use(TextureUnit.Texture1);
        _shader.Use();

        // Нарешті маємо матрицю моделі
        var model = Matrix4.Identity * Ma-
trix4.CreateRotationX((float)MathHelper.DegreesToRadians(_time));

        // Потім ми передаємо всі ці матриці до вершинного шейдера.
        _shader.SetMatrix4("model", model);
        _shader.SetMatrix4("view", _view);
        _shader.SetMatrix4("projection", _projection);

        GL.DrawElements(PrimitiveType.Triangles, _indices.Length, DrawElement-
sType.UnsignedInt, 0);
```

```
        SwapBuffers();
    }

    protected override void OnUpdateFrame(FrameEventArgs e)
    {
        base.OnUpdateFrame(e);

        var input = KeyboardState;

        if (input.IsKeyDown(Keys.Escape))
        {
            Close();
        }
    }

    protected override void OnResize(ResizeEventArgs e)
    {
        base.OnResize(e);

        GL.Viewport(0, 0, Size.X, Size.Y);
    }
}
```

shader.vert

```
#version 330 core

layout(location = 0) in vec3 aPosition;

layout(location = 1) in vec2 aTexCoord;

out vec2 texCoord;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main(void)
{
    texCoord = aTexCoord;

    gl_Position = vec4(aPosition, 1.0) * model * view * projection;
}
```


Madness

7. Камера

Window.cs

```
// створимо камеру
// Насправді ми не можемо рухати камеру, але ми фактично рухаємо прямокутник.
public class Window : GameWindow
{
    private readonly float[] _vertices =
    {
            // позиція              координати текстури
             0.5f,  0.5f, 0.0f,      1.0f, 1.0f,
             0.5f, -0.5f, 0.0f,      1.0f, 0.0f,
            -0.5f, -0.5f, 0.0f,      0.0f, 0.0f,
            -0.5f,  0.5f, 0.0f,      0.0f, 1.0f
        };

    private readonly uint[] _indices =
    {
            0, 1, 3,
            1, 2, 3
        };

    private int _elementBufferObject;

    private int _vertexBufferObject;

    private int _vertexArrayObject;

    private Shader _shader;

    private Texture _texture;

    private Texture _texture2;

    // Матриці перегляду та проекції видалено, оскільки вони більше тут не потрібні.
    // Тепер вони в класі камера

    // потрібен екземпляр нового класу камери, щоб він міг керувати кодом матриці пе-
регляду та проекції.
    // також потрібне логічне значення true, щоб визначити, чи була миша переміщена впер-
ше.
    // додаємо останню позицію миші, щоб ми могли легко обчислити зсув миші.
    private Camera _camera;

    private bool _firstMove = true;

    private Vector2 _lastPos;

    private double _time;

    public Window(GameWindowSettings gameWindowSettings, NativeWindowSettings nativeWin-
dowSettings)
        : base(gameWindowSettings, nativeWindowSettings)
    {
    }

    protected override void OnLoad()
    {
        base.OnLoad();

        GL.ClearColor(0.2f, 0.3f, 0.3f, 1.0f);

        GL.Enable(EnableCap.DepthTest);

        _vertexArrayObject = GL.GenVertexArray();
```

```
        GL.BindVertexArray(_vertexArrayObject);

        _vertexBufferObject = GL.GenBuffer();
        GL.BindBuffer(BufferTarget.ArrayBuffer, _vertexBufferObject);
        GL.BufferData(BufferTarget.ArrayBuffer, _vertices.Length * sizeof(float),
_vertices, BufferUsageHint.StaticDraw);

        _elementBufferObject = GL.GenBuffer();
        GL.BindBuffer(BufferTarget.ElementArrayBuffer, _elementBufferObject);
        GL.BufferData(BufferTarget.ElementArrayBuffer, _indices.Length * sizeof(uint),
_indices, BufferUsageHint.StaticDraw);

        _shader = new Shader("Shaders/shader.vert", "Shaders/fragshader.frag");
        _shader.Use();

        var vertexLocation = _shader.GetAttribLocation("aPosition");
        GL.EnableVertexAttribArray(vertexLocation);
        GL.VertexAttribPointer(vertexLocation, 3, VertexAttribPointerType.Float, false, 5
* sizeof(float), 0);

        var texCoordLocation = _shader.GetAttribLocation("aTexCoord");
        GL.EnableVertexAttribArray(texCoordLocation);
        GL.VertexAttribPointer(texCoordLocation, 2, VertexAttribPointerType.Float, false,
5 * sizeof(float), 3 * sizeof(float));

        _texture = Texture.LoadFromFile("Resources/container.png");
        _texture.Use(TextureUnit.Texture0);

        _texture2 = Texture.LoadFromFile("Resources/awesomeface.png");
        _texture2.Use(TextureUnit.Texture1);

        _shader.SetInt("texture0", 0);
        _shader.SetInt("texture1", 1);

        // Ми ініціалізуємо камеру так, щоб вона була на 3 одиниці позаду від прямокутни-
ка.
        // Ми також надаємо йому належне співвідношення сторін.
        _camera = new Camera(Vector3.UnitZ * 3, Size.X / (float)Size.Y);

        // захоплюємо курсор миші і робимо його невидимим
        CursorState = CursorState.Grabbed;
    }

    protected override void OnRenderFrame(FrameEventArgs e)
    {
        base.OnRenderFrame(e);

        _time += 4.0 * e.Time;

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

        GL.BindVertexArray(_vertexArrayObject);

        _texture.Use(TextureUnit.Texture0);
        _texture2.Use(TextureUnit.Texture1);
        _shader.Use();

        var model = Matrix4.Identity * Ma-
trix4.CreateRotationX((float)MathHelper.DegreesToRadians(_time));
        _shader.SetMatrix4("model", model);
        _shader.SetMatrix4("view", _camera.GetViewMatrix());
        _shader.SetMatrix4("projection", _camera.GetProjectionMatrix());

        GL.DrawElements(PrimitiveType.Triangles, _indices.Length, DrawElement-
sType.UnsignedInt, 0);
```

```
        SwapBuffers();
    }

    protected override void OnUpdateFrame(FrameEventArgs e)
    {
        base.OnUpdateFrame(e);

        if (!IsFocused) // перевірка чи вікно у фокусі
        {
            return;
        }

        var input = KeyboardState;

        if (input.IsKeyDown(Keys.Escape))
        {
            Close();
        }

        const float cameraSpeed = 1.5f;
        const float sensitivity = 0.2f;

        if (input.IsKeyDown(Keys.W))
        {
            _camera.Position += _camera.Front * cameraSpeed * (float)e.Time; // вперед
        }

        if (input.IsKeyDown(Keys.S))
        {
            _camera.Position -= _camera.Front * cameraSpeed * (float)e.Time; // назад
        }
        if (input.IsKeyDown(Keys.A))
        {
            _camera.Position -= _camera.Right * cameraSpeed * (float)e.Time; // ліво
        }
        if (input.IsKeyDown(Keys.D))
        {
            _camera.Position += _camera.Right * cameraSpeed * (float)e.Time; // право
        }
        if (input.IsKeyDown(Keys.Space))
        {
            _camera.Position += _camera.Up * cameraSpeed * (float)e.Time; // вверх
        }
        if (input.IsKeyDown(Keys.LeftShift))
        {
            _camera.Position -= _camera.Up * cameraSpeed * (float)e.Time; // вниз
        }

        // отримаємо стан миші
        var mouse = MouseState;

        if (_firstMove) // Для цієї змінної bool спочатку встановлено значення true.
        {
            _lastPos = new Vector2(mouse.X, mouse.Y);
            _firstMove = false;
        }
        else
        {
            // обчислення зміщення до позиції миші
            var deltaX = mouse.X - _lastPos.X;
            var deltaY = mouse.Y - _lastPos.Y;
            _lastPos = new Vector2(mouse.X, mouse.Y);

            // обрахуємо крок камери
```

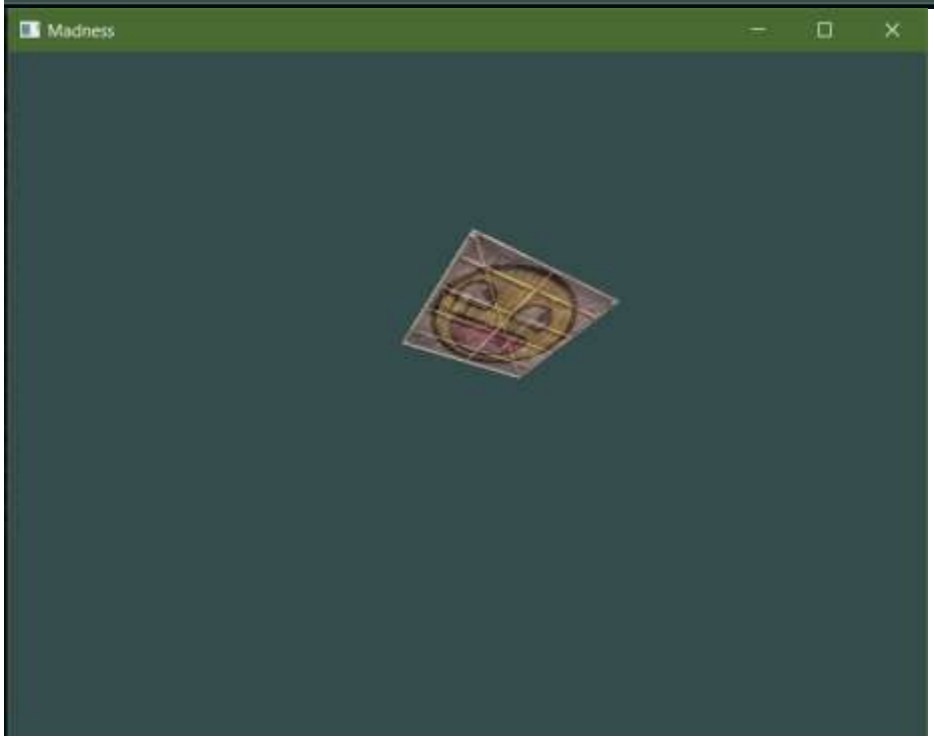
```
            _camera.Yaw += deltaX * sensitivity;
            _camera.Pitch -= deltaY * sensitivity; // перевертаємо оскільки Y йде в низ
        }
    }

    // Для функції коліщатка миші застосуємо масштабування камери.

    protected override void OnMouseWheel(MouseWheelEventArgs e)
    {
        base.OnMouseWheel(e);

        _camera.Fov -= e.OffsetY;
    }

    protected override void OnResize(ResizeEventArgs e)
    {
        base.OnResize(e);

        GL.Viewport(0, 0, Size.X, Size.Y);
        // оновимо співвідношеннят сторін при зміні вікна
        _camera.AspectRatio = Size.X / (float)Size.Y;
    }
}
```

Camera.cs

```
// налаштуємо клас камери
public class Camera
{
    // створимо вектори напрямку руху камери
    private Vector3 _front = -Vector3.UnitZ;

    private Vector3 _up = Vector3.UnitY;

    private Vector3 _right = Vector3.UnitX;

    // обертання по X
    private float _pitch;

    // обертання по Y
    private float _yaw = -MathHelper.PiOver2; // костиль для повороту на 90 градусів
ліворуч

    // поле зору камери
    private float _fov = MathHelper.PiOver2;

    public Camera(Vector3 position, float aspectRatio)
    {
        Position = position;
        AspectRatio = aspectRatio;
    }

    // позиція камери
    public Vector3 Position { get; set; }

    // це просто співвідношення сторін вікна перегляду, яке використовується для матриці
проекції.
    public float AspectRatio { private get; set; }

    public Vector3 Front => _front;

    public Vector3 Up => _up;

    public Vector3 Right => _right;
```

```
    // перетворюємо градуси в радіани, як тільки встановлюється властивість для покращення продуктивності.
    public float Pitch
    {
        get => MathHelper.RadiansToDegrees(_pitch);
        set
        {
            // фіксуємо значення кроку між -89 і 89, щоб запобігти перекиданню камери догори дном, і багів при використанні кутів ейлера
            var angle = MathHelper.Clamp(value, -89f, 89f);
            _pitch = MathHelper.DegreesToRadians(angle);
            UpdateVectors();
        }
    }

    // перетворюємо градуси в радіани, як тільки встановлюється властивість для покращення продуктивності.
    public float Yaw
    {
        get => MathHelper.RadiansToDegrees(_yaw);
        set
        {
            _yaw = MathHelper.DegreesToRadians(value);
            UpdateVectors();
        }
    }

    // поле зору камери (FOV)
    // перетворюємо градуси в радіани, як тільки встановлюється властивість для покращення продуктивності.
    public float Fov
    {
        get => MathHelper.RadiansToDegrees(_fov);
        set
        {
            var angle = MathHelper.Clamp(value, 1f, 90f);
            _fov = MathHelper.DegreesToRadians(angle);
        }
    }

    // Отримаємо матрицю перегляду за допомогою функції LookAt,
    public Matrix4 GetViewMatrix()
    {
        return Matrix4.LookAt(Position, Position + _front, _up);
    }

    // Отримаємо матрицю проекції
    public Matrix4 GetProjectionMatrix()
    {
        return Matrix4.CreatePerspectiveFieldOfView(_fov, AspectRatio, 0.01f, 100f);
    }

    // Ця функція оновить векторні вершини
    private void UpdateVectors()
    {
        // Спочатку передня матриця обчислюється за допомогою базової тригонометрії.
        _front.X = MathF.Cos(_pitch) * MathF.Cos(_yaw);
        _front.Y = MathF.Sin(_pitch);
        _front.Z = MathF.Cos(_pitch) * MathF.Sin(_yaw);

        // перевірка чи нормалізовані вектори, щоб не було гкхм.. цікавих результатів
        _front = Vector3.Normalize(_front);

        // Обчислення правого та верхнього векторів за допомогою перехресного добутку.
        _right = Vector3.Normalize(Vector3.Cross(_front, Vector3.UnitY));
```

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



8.Опрацювання матеріалів і освітлення

Window.cs

```
public class Window : GameWindow
{
    private readonly float[] _vertices =
    {
        //  Позиція            Нормалі             текстура
        -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[] _cubePositions =
{
        new Vector3(0.0f, 0.0f, 0.0f),
        new Vector3(2.0f, 5.0f, -15.0f),
        new Vector3(-1.5f, -2.2f, -2.5f),
        new Vector3(-3.8f, -2.0f, -12.3f),
        new Vector3(2.4f, -0.4f, -3.5f),
        new Vector3(-1.7f, 3.0f, -7.5f),
        new Vector3(1.3f, -2.0f, -2.5f),
        new Vector3(1.5f, 2.0f, -2.5f),
        new Vector3(1.5f, 0.2f, -1.5f),
        new Vector3(-1.3f, 1.0f, -1.5f)
    };


private readonly Vector3[] _pointLightPositions =
{
        new Vector3(0.7f, 0.2f, 2.0f),
        new Vector3(2.3f, -3.3f, -4.0f),
        new Vector3(-4.0f, 2.0f, -12.0f),
        new Vector3(0.0f, 0.0f, -3.0f)
    };

private int _vertexBufferObject;

private int _vaoModel;

private int _vaoLamp;

private Shader _lampShader;

private Shader _lightingShader;

private Texture _diffuseMap;

private Texture _specularMap;

private Camera _camera;
```

```
    private bool _firstMove = true;

    private Vector2 _lastPos;

    public Window(GameWindowSettings gameWindowSettings,
NativeWindowSettings nativeWindowSettings)
        : base(gameWindowSettings, nativeWindowSettings)
    {
    }

    protected override void OnLoad()
    {
        base.OnLoad();

        GL.ClearColor(0.2f, 0.3f, 0.3f, 1.0f);

        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("Shaders/shader.vert",
"Shaders/lighting.frag");
        _lampShader = new Shader("Shaders/shader.vert",
"Shaders/fragshader.frag");

        {
            _vaoModel = GL.GenVertexArray();
            GL.BindVertexArray(_vaoModel);

            var positionLocation = _lightingShader.GetAttribLocation("aPos");
            GL.EnableVertexAttribArray(positionLocation);
            GL.VertexAttribPointer(positionLocation, 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);

        var positionLocation = _lampShader.GetAttribLocation("aPos");
        GL.EnableVertexAttribArray(positionLocation);
        GL.VertexAttribPointer(positionLocation, 3,
VertexAttribPointerType.Float, false, 8 * sizeof(float), 0);
    }

    _diffuseMap = Texture.LoadFromFile("Resources/container2.png");
    _specularMap =
Texture.LoadFromFile("Resources/container2_specular.png");

    _camera = new Camera(Vector3.UnitZ * 3, Size.X / (float)Size.Y);

    CursorState = CursorState.Grabbed;
  }

  protected override void OnRenderFrame(FrameEventArgs e)
  {
    base.OnRenderFrame(e);

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

    GL.BindVertexArray(_vaoModel);

    _diffuseMap.Use(TextureUnit.Texture0);
    _specularMap.Use(TextureUnit.Texture1);
    _lightingShader.Use();

    _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.SetVector3("material.specular", new Vector3(0.5f, 0.5f,
0.5f));
        _lightingShader.SetFloat("material.shininess", 32.0f);


        _lightingShader.SetVector3("dirLight.direction", new Vector3(-0.2f, -1.0f, -
0.3f));
        _lightingShader.SetVector3("dirLight.ambient", new Vector3(0.05f, 0.05f,
0.05f));
        _lightingShader.SetVector3("dirLight.diffuse", new Vector3(0.4f, 0.4f, 0.4f));
        _lightingShader.SetVector3("dirLight.specular", new Vector3(0.5f, 0.5f,
0.5f));

        // Point lights
        for (int i = 0; i < _pointLightPositions.Length; i++)
        {
            _lightingShader.SetVector3($"pointLights[{i}].position",
_pointLightPositions[i]);
            _lightingShader.SetVector3($"pointLights[{i}].ambient", new
Vector3(0.05f, 0.05f, 0.05f));
            _lightingShader.SetVector3($"pointLights[{i}].diffuse", new Vector3(0.8f,
0.8f, 0.8f));
            _lightingShader.SetVector3($"pointLights[{i}].specular", new Vector3(1.0f,
1.0f, 1.0f));
            _lightingShader.SetFloat($"pointLights[{i}].constant", 1.0f);
            _lightingShader.SetFloat($"pointLights[{i}].linear", 0.09f);
            _lightingShader.SetFloat($"pointLights[{i}].quadratic", 0.032f);
        }

        // Spot light
        _lightingShader.SetVector3("spotLight.position", _camera.Position);
        _lightingShader.SetVector3("spotLight.direction", _camera.Front);
        _lightingShader.SetVector3("spotLight.ambient", new Vector3(0.0f, 0.0f,
0.0f));
        _lightingShader.SetVector3("spotLight.diffuse", new Vector3(1.0f, 1.0f,
1.0f));
        _lightingShader.SetVector3("spotLight.specular", new Vector3(1.0f, 1.0f,
1.0f));
        _lightingShader.SetFloat("spotLight.constant", 1.0f);
        _lightingShader.SetFloat("spotLight.linear", 0.09f);
        _lightingShader.SetFloat("spotLight.quadratic", 0.032f);
```

```
        _lightingShader.SetFloat("spotLight.cutOff",
MathF.Cos(MathHelper.DegreesToRadians(12.5f)));
        _lightingShader.SetFloat("spotLight.outerCutOff",
MathF.Cos(MathHelper.DegreesToRadians(17.5f)));

        for (int i = 0; i < _cubePositions.Length; i++)
        {
            Matrix4 model = Matrix4.CreateTranslation(_cubePositions[i]);
            float angle = 20.0f * i;
            model = model * Matrix4.CreateFromAxisAngle(new Vector3(1.0f, 0.3f,
0.5f), angle);
            _lightingShader.SetMatrix4("model", model);

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

        GL.BindVertexArray(_vaoLamp);

        _lampShader.Use();

        _lampShader.SetMatrix4("view", _camera.GetViewMatrix());
        _lampShader.SetMatrix4("projection", _camera.GetProjectionMatrix());

        for (int i = 0; i < _pointLightPositions.Length; i++)
        {
            Matrix4 lampMatrix = Matrix4.CreateScale(0.2f);
            lampMatrix = lampMatrix *
Matrix4.CreateTranslation(_pointLightPositions[i]);

            _lampShader.SetMatrix4("model", lampMatrix);

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

        SwapBuffers();
    }

    protected override void OnUpdateFrame(FrameEventArgs e)
    {
        base.OnUpdateFrame(e);

        if (!IsFocused)
        {
            return;
```

```
}

var input = KeyboardState;

if (input.IsKeyDown(Keys.Escape))
{
   Close();
}

const float cameraSpeed = 1.5f;
const float sensitivity = 0.2f;

if (input.IsKeyDown(Keys.W))
{
   _camera.Position += _camera.Front * cameraSpeed * (float)e.Time;
}
if (input.IsKeyDown(Keys.S))
{
   _camera.Position -= _camera.Front * cameraSpeed * (float)e.Time;
}
if (input.IsKeyDown(Keys.A))
{
   _camera.Position -= _camera.Right * cameraSpeed * (float)e.Time;
}
if (input.IsKeyDown(Keys.D))
{
   _camera.Position += _camera.Right * cameraSpeed * (float)e.Time;
}
if (input.IsKeyDown(Keys.Space))
{
   _camera.Position += _camera.Up * cameraSpeed * (float)e.Time;
}
if (input.IsKeyDown(Keys.LeftShift))
{
   _camera.Position -= _camera.Up * cameraSpeed * (float)e.Time;
}

var mouse = MouseState;

if (_firstMove)
{
   _lastPos = new Vector2(mouse.X, mouse.Y);
   _firstMove = false;
}
```

```
        else
        {
            var deltaX = mouse.X - _lastPos.X;
            var deltaY = mouse.Y - _lastPos.Y;
            _lastPos = new Vector2(mouse.X, mouse.Y);

            _camera.Yaw += deltaX * sensitivity;
            _camera.Pitch -= deltaY * sensitivity;
        }
    }

    protected override void OnMouseWheel(MouseWheelEventArgs e)
    {
        base.OnMouseWheel(e);

        _camera.Fov -= e.OffsetY;
    }

    protected override void OnResize(ResizeEventArgs e)
    {
        base.OnResize(e);

        GL.Viewport(0, 0, Size.X, Size.Y);
        _camera.AspectRatio = Size.X / (float)Size.Y;
    }
}
```

Shader.vert

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

out vec3 Normal;
out vec3 FragPos;
out vec2 TexCoords;

void main()
{
    gl_Position = vec4(aPos, 1.0) * model * view * projection;
    FragPos = vec3(vec4(aPos, 1.0) * model);
    Normal = aNormal * mat3(transpose(inverse(model)));
    TexCoords = aTexCoords;
}
```

Fragshader.frag

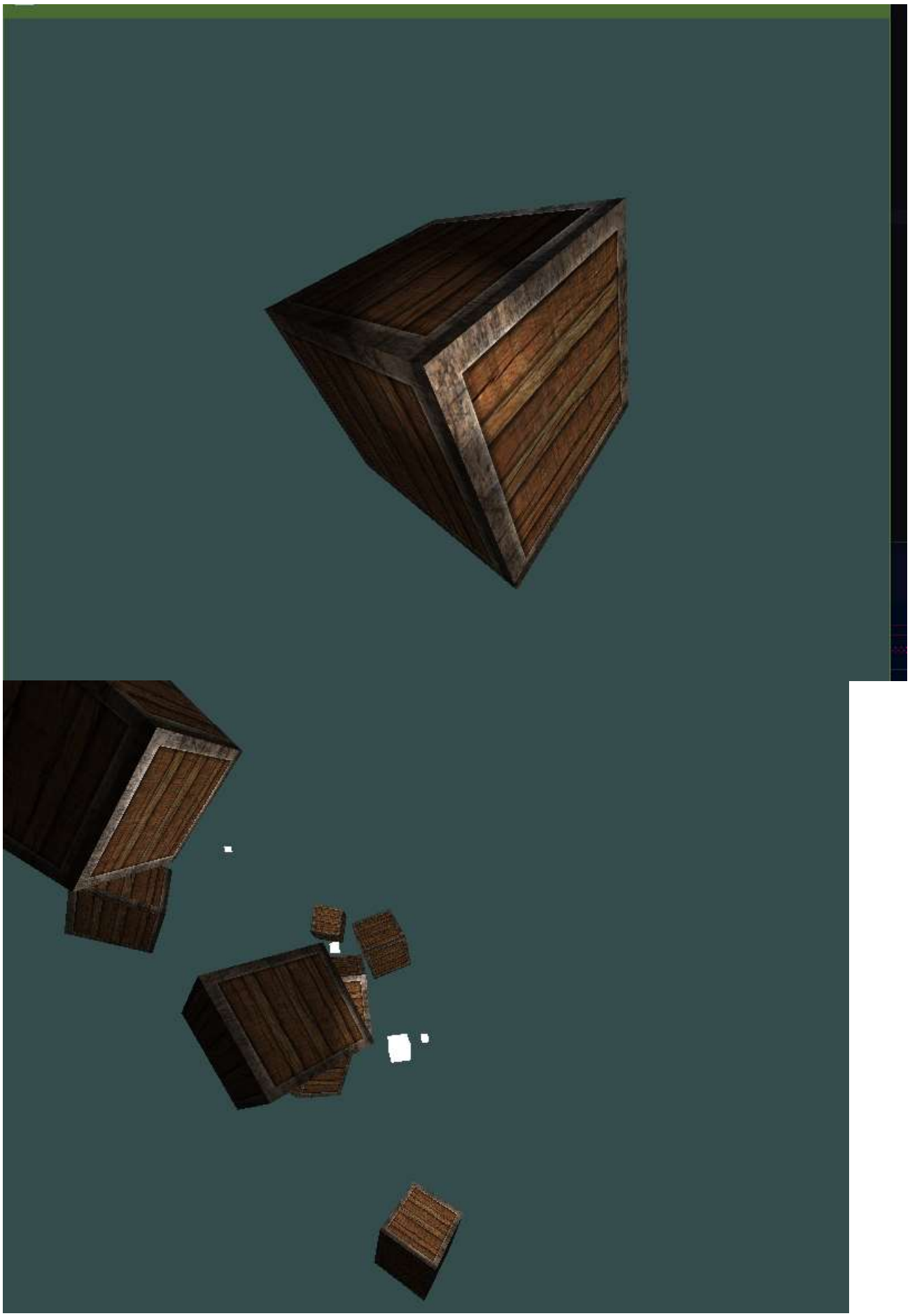
```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(1.0); // set all 4 vector values to 1.0
}
```

Lighting.frag

```
#version 330 core

struct Material {
    sampler2D diffuse;
    sampler2D specular;
    float     shininess;
};

struct DirLight {
    vec3 direction;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};
uniform DirLight dirLight;

struct PointLight {
    vec3 position;

    float constant;
    float linear;
    float quadratic;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};

#define NR_POINT_LIGHTS 4
uniform PointLight pointLights[NR_POINT_LIGHTS];

struct SpotLight{
    vec3  position;
    vec3  direction;
    float cutOff;
    float outerCutOff;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;

    float constant;
    float linear;
    float quadratic;
};
uniform SpotLight spotLight;

uniform Material material;
uniform vec3 viewPos;

out vec4 FragColor;

in vec3 Normal;
in vec3 FragPos;
```

```
in vec2 TexCoords;


vec3 CalcDirLight(DirLight light, vec3 normal, vec3 viewDir);
vec3 CalcPointLight(PointLight light, vec3 normal, vec3 fragPos, vec3 viewDir);
vec3 CalcSpotLight(SpotLight light, vec3 normal, vec3 fragPos, vec3 viewDir);

void main()
{
    vec3 norm = normalize(Normal);
    vec3 viewDir = normalize(viewPos - FragPos);

    vec3 result = CalcDirLight(dirLight, norm, viewDir);

    for(int i = 0; i < NR_POINT_LIGHTS; i++)
        result += CalcPointLight(pointLights[i], norm, FragPos, viewDir);

    result += CalcSpotLight(spotLight, norm, FragPos, viewDir);

    FragColor = vec4(result, 1.0);
}

vec3 CalcDirLight(DirLight light, vec3 normal, vec3 viewDir)
{
    vec3 lightDir = normalize(-light.direction);

    float diff = max(dot(normal, lightDir), 0.0);

    vec3 reflectDir = reflect(-lightDir, normal);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);

    vec3 ambient  = light.ambient  * vec3(texture(material.diffuse, TexCoords));
    vec3 diffuse  = light.diffuse  * diff * vec3(texture(material.diffuse, TexCoords));
    vec3 specular = light.specular * spec * vec3(texture(material.specular, TexCoords));
    return (ambient + diffuse + specular);
}

vec3 CalcPointLight(PointLight light, vec3 normal, vec3 fragPos, vec3 viewDir)
{
    vec3 lightDir = normalize(light.position - fragPos);

    float diff = max(dot(normal, lightDir), 0.0);

    vec3 reflectDir = reflect(-lightDir, normal);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);

    float distance    = length(light.position - fragPos);
    float attenuation = 1.0 / (light.constant + light.linear * distance +
    light.quadratic * (distance * distance));

    vec3 ambient  = light.ambient  * vec3(texture(material.diffuse, TexCoords));
    vec3 diffuse  = light.diffuse  * diff * vec3(texture(material.diffuse, TexCoords));
    vec3 specular = light.specular * spec * vec3(texture(material.specular, TexCoords));
    ambient  *= attenuation;
    diffuse  *= attenuation;
    specular *= attenuation;
    return (ambient + diffuse + specular);
}
vec3 CalcSpotLight(SpotLight light, vec3 normal, vec3 fragPos, vec3 viewDir)
{


    vec3 lightDir = normalize(light.position - FragPos);
    float diff = max(dot(normal, lightDir), 0.0);
```

```
    vec3 reflectDir = reflect(-lightDir, normal);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);


    float distance    = length(light.position - FragPos);
    float attenuation = 1.0 / (light.constant + light.linear * distance +
    light.quadratic * (distance * distance));


    float theta     = dot(lightDir, normalize(-light.direction));
    float epsilon   = light.cutOff - light.outerCutOff;
    float intensity = clamp((theta - light.outerCutOff) / epsilon, 0.0, 1.0);


    vec3 ambient = light.ambient * vec3(texture(material.diffuse, TexCoords));
    vec3 diffuse = light.diffuse * diff * vec3(texture(material.diffuse, TexCoords));
    vec3 specular = light.specular * spec * vec3(texture(material.specular, TexCoords));
    ambient  *= attenuation;
    diffuse  *= attenuation * intensity;
    specular *= attenuation * intensity;
    return (ambient + diffuse + specular);
}
```



Код завантажено до репозиторію: https://github.com/Gw1xo/openTK_Practic.git
Завдання розкидані по коммітам

Висновок: Опрацьовано основи роботи з openGL. Проведена робота з відмальовки примітивів роботи з буфферами, шейдерами , трансформаціями, роботи з камерою, текстурами, матеріалами, базовим і розширеним освітленням