МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»

НГТУ НЭТИ | **Факультет прикладной математики и информатики**

Кафедра прикладной математики

Практическое задание №2
по дисциплине «Компьютерная графика»

Трёхмерная визуализация в режиме реального времени

Группа: ПМ-02 ДАНЧЕНКО ИВАН
Вариант: 4

Преподаватели: ЗАДОРОЖНЫЙ А.Г.

Новосибирск, 2023

Цель работы

Ознакомиться со средствами предоставления полномочий наС использование схем и баз данных и таблиц и основами работы с внешними базами данных.

Средства реализации приложения

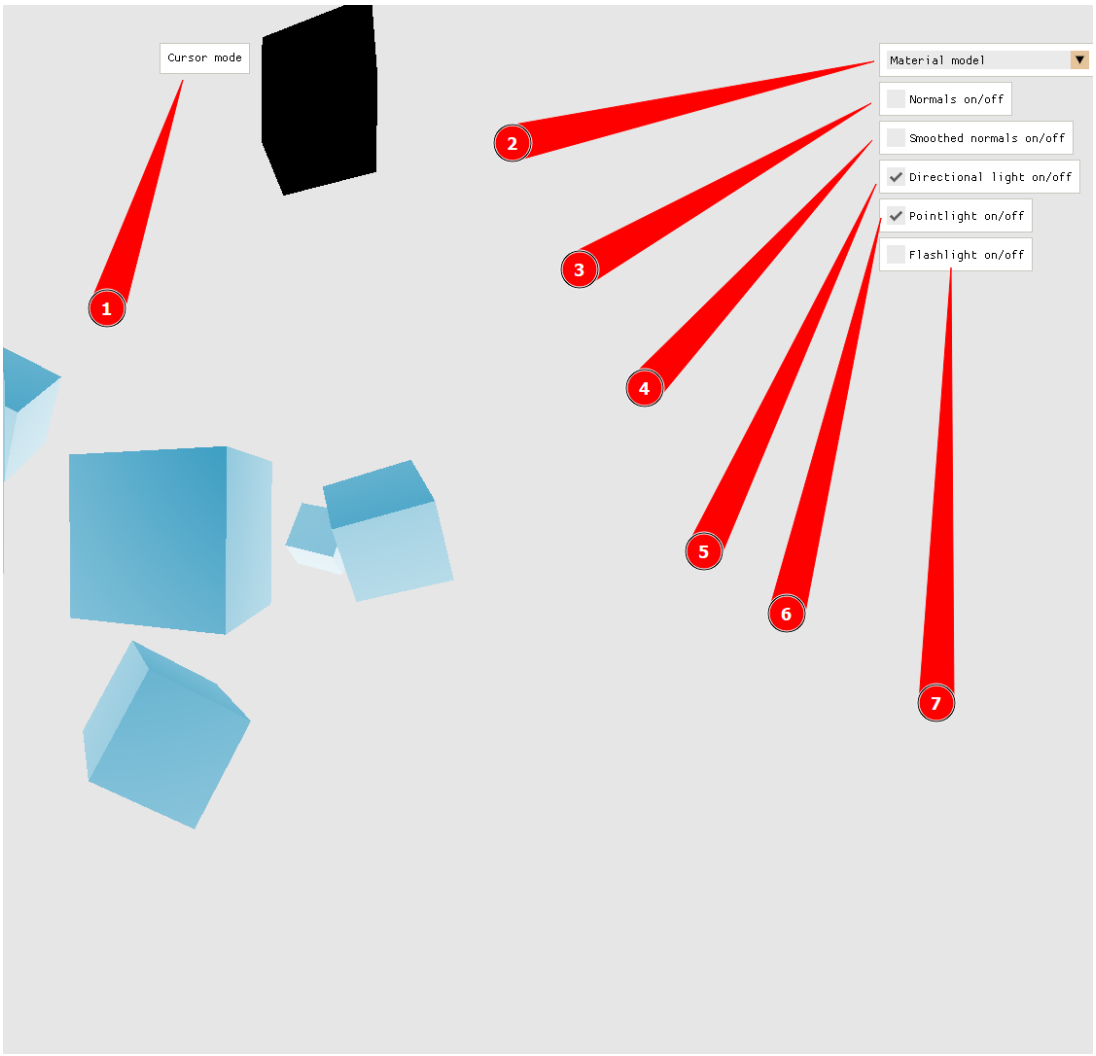
Основной язык - C#, тулкит для OpenGL - библиотека OpenTK, графический интерфейс реализовывался с помощью ImGUI.

Возможности приложения

- Есть возможность переключаться между перспективной и ортографической камерой:
  - **P** - режим перспективной камеры
  - **O** - режим ортографической камеры
- Есть 2 режима, между которыми пользователь переключается с помощью кнопок:
  - **M** - режим передвижения
  - **N** - режим управлением курсора.
- В режиме передвижения пользователь может перемещаться с помощью кнопок:
  - **W**, **A**, **S**, **D** - нам всем известные клавиши.
  - **Space/Shift** - перемещение вверх/вниз
  - **Mousewheel up/down** - снижение/уменьшение FOV
- В режиме курсора пользователь работает с интерфейсом программы.

Немного про интерфейс

1. Поле является подсказкой для пользователя в каком режиме программы он сейчас находится (в режиме перемещения или работы курсора)
2. Выбор типа объекта (материал, текстура, рамка фигуры)
3. Включение/выключение показа нормалей
4. Включение/выключение сглаженных нормалей
5. Включение/выключение прямого источника света
6. Включение/выключение лампы (точечного источника света)
7. Включение/выключение фонарика


Cursor mode
Material model
Normals on/off
Smoothed normals on/off
Directional light on/off
Pointlight on/off
Flashlight on/off
1
2
3
4
5
6
7

Пример работы программы

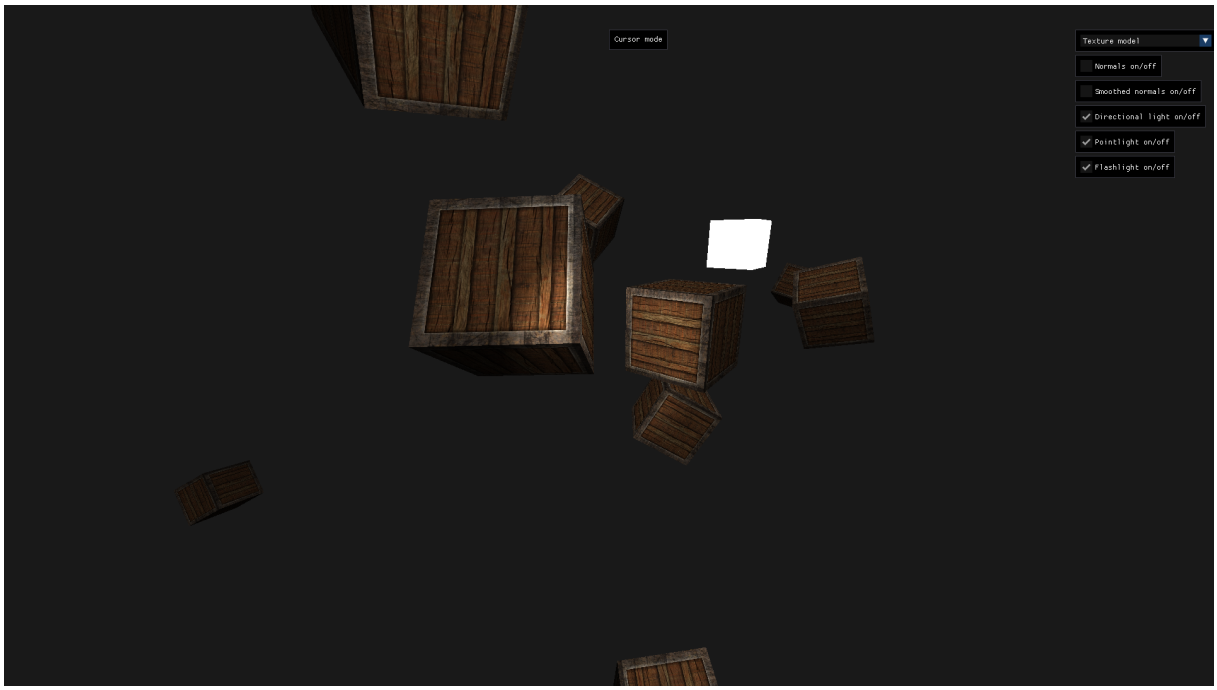


Рисунок 1: Включены все источники света

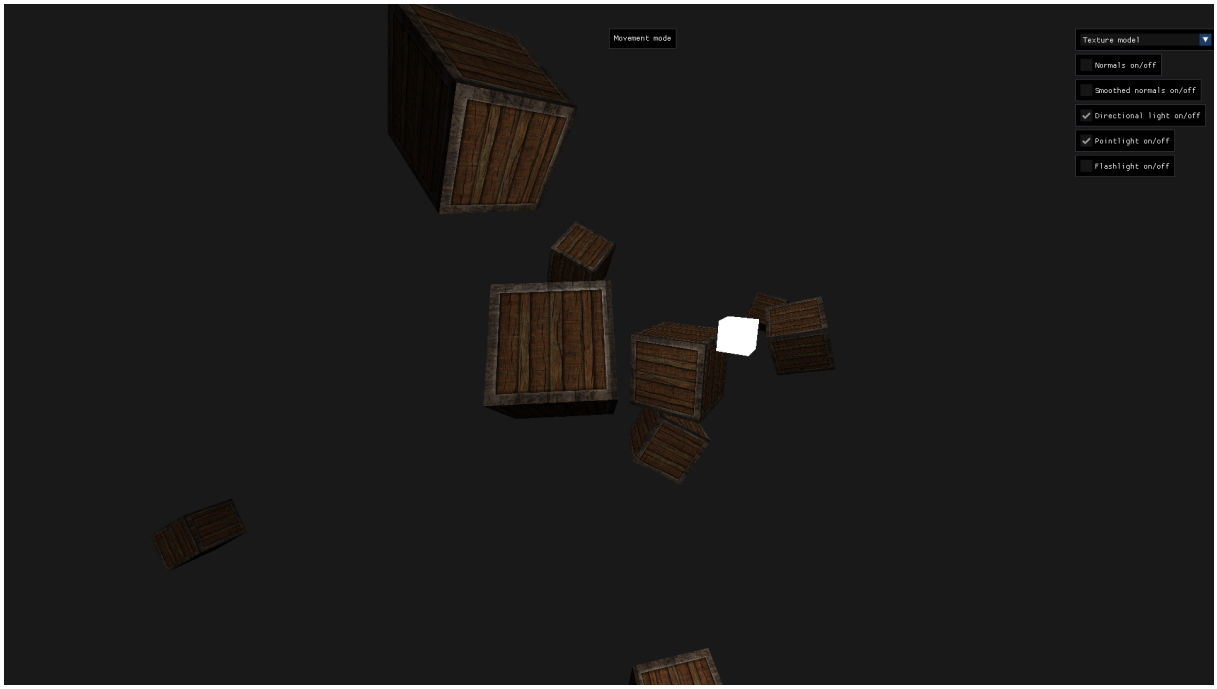


Рисунок 2: Включен прямой свет и лампа

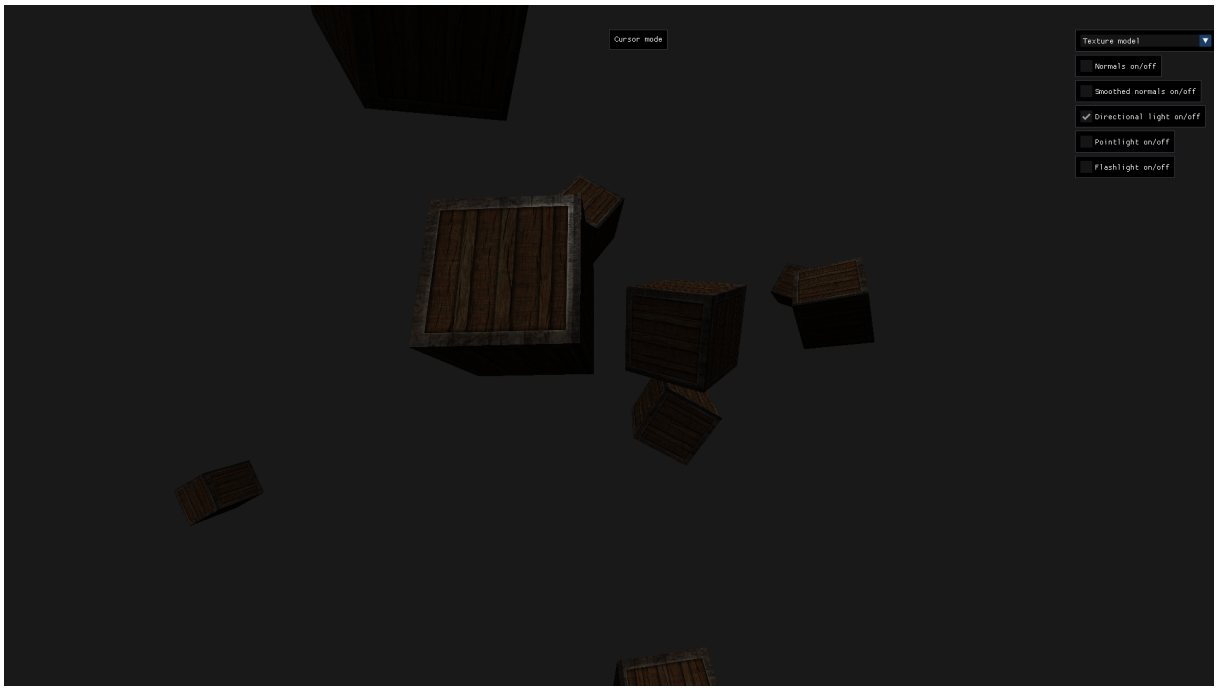


Рисунок 3: Включен только прямой свет

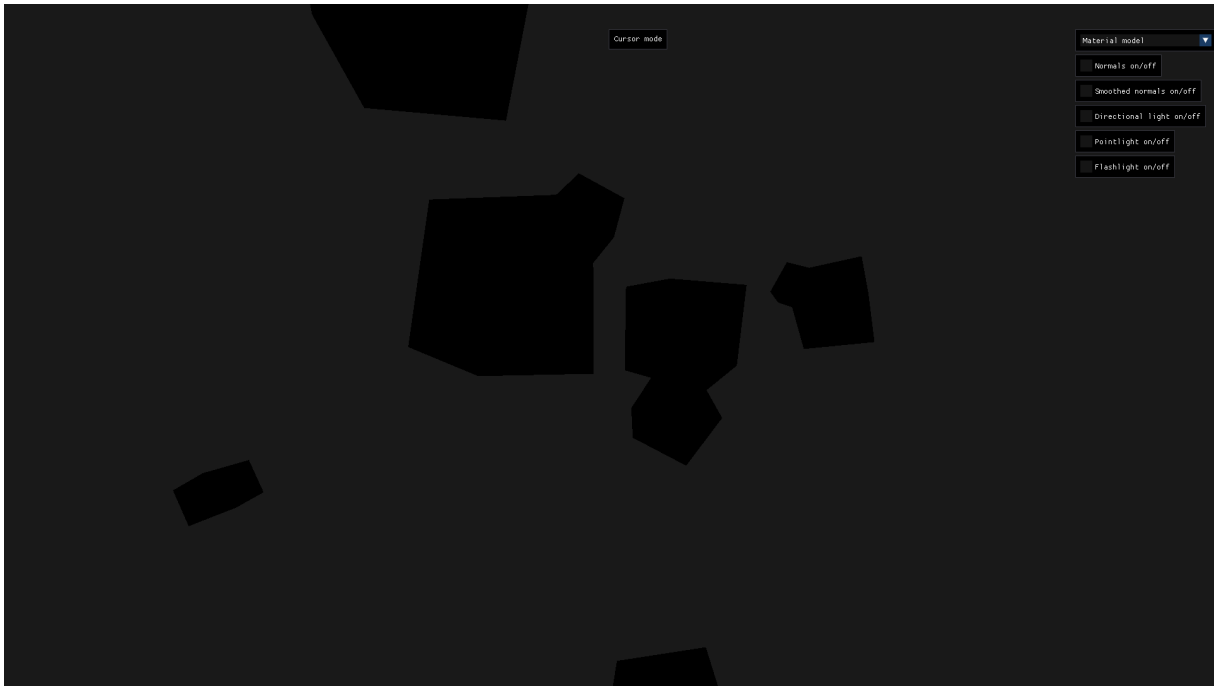


Рисунок 4: Весь свет выключен

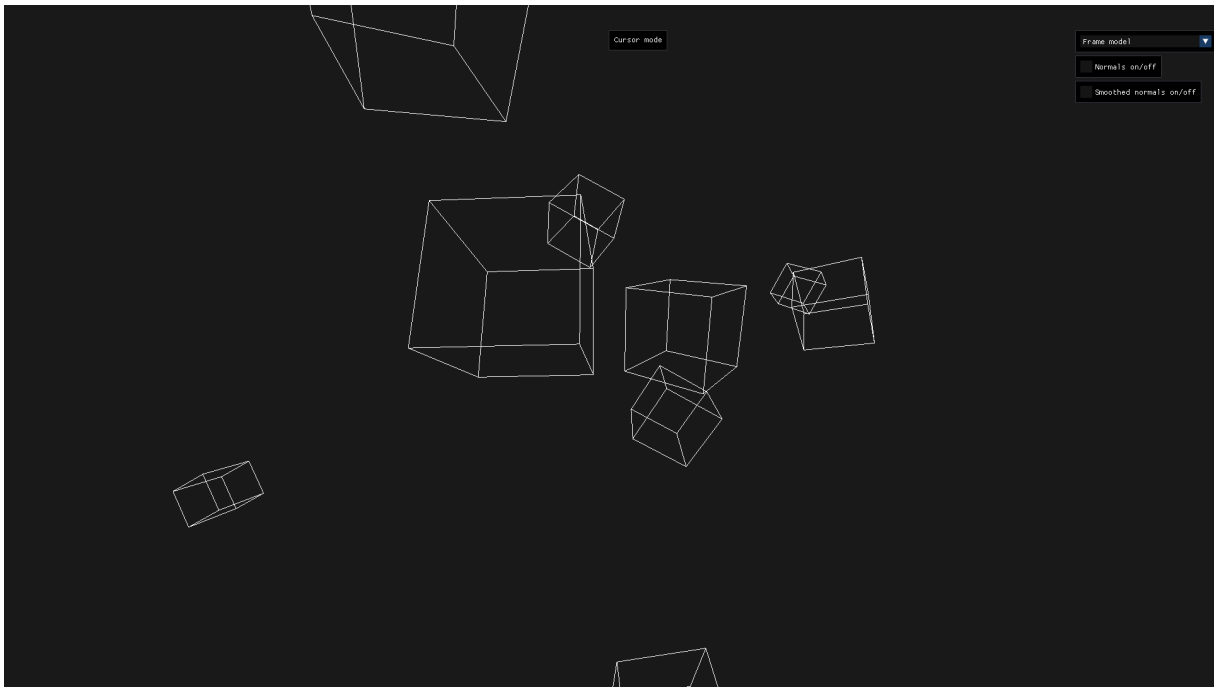


Рисунок 5: Показ каркаса моделей

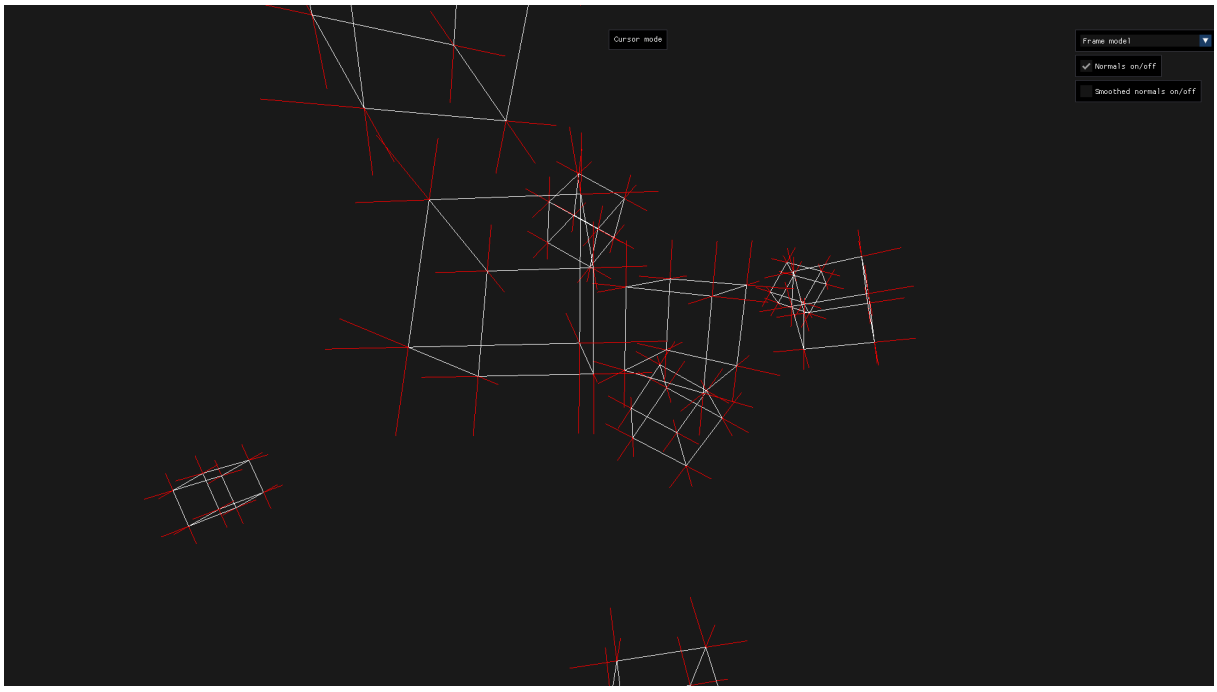


Рисунок 6: Показ нормалей модели

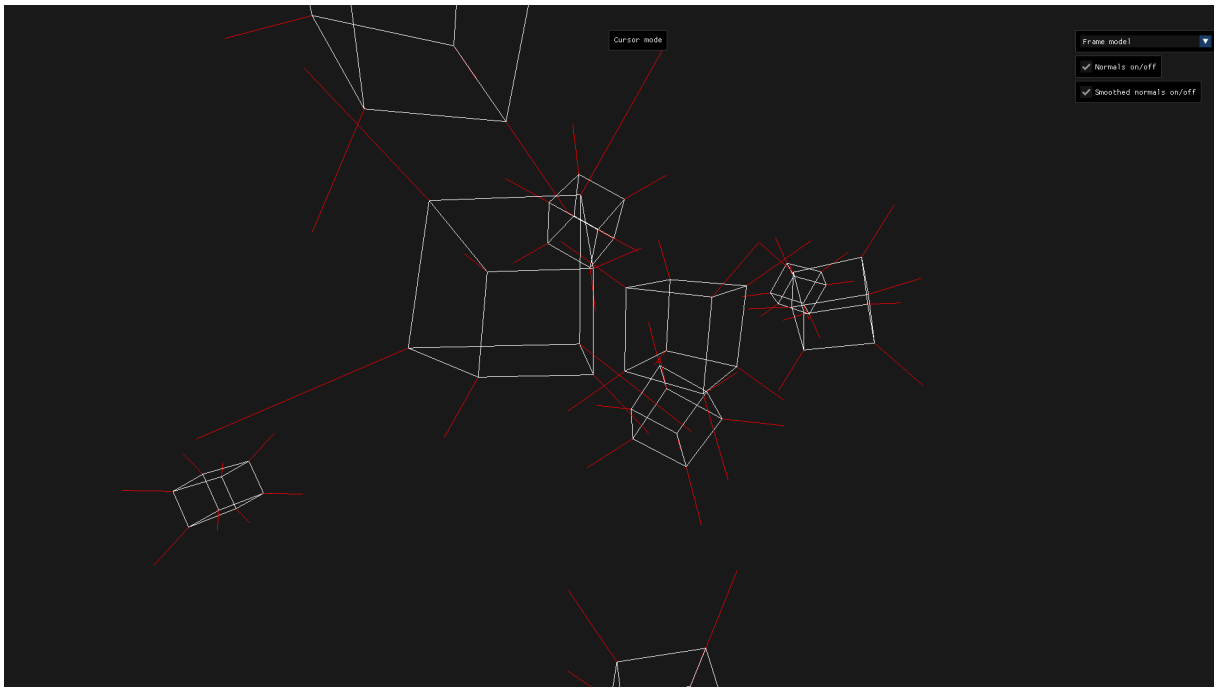


Рисунок 7: Показ сглаженных нормалей модели

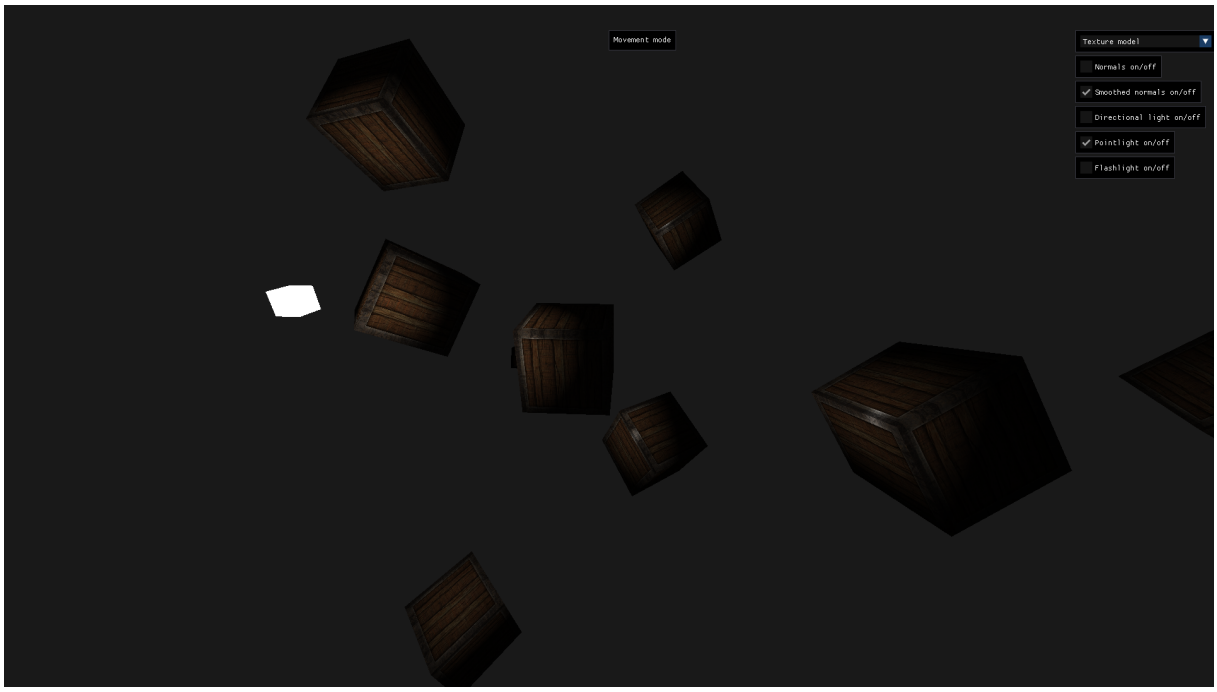


Рисунок 8: Пример работы освещения при сглаженных нормалях

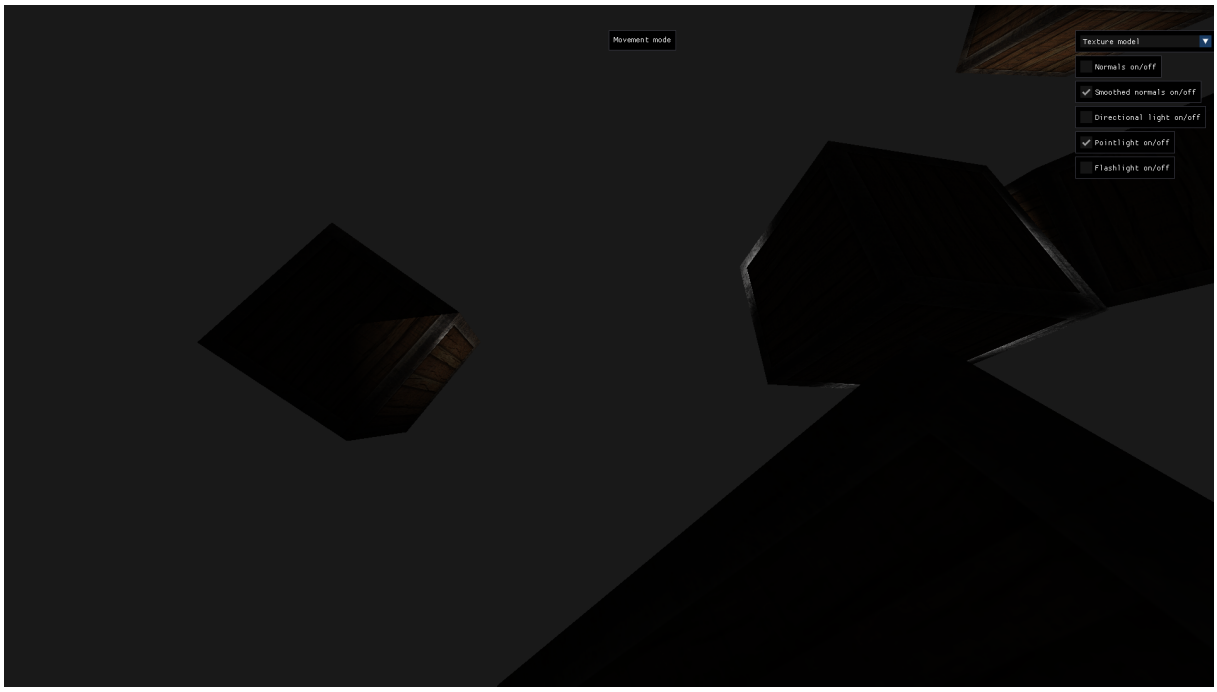


Рисунок 9: Пример почему сглаженные нормали это плохо (смотреть рисунок 10)

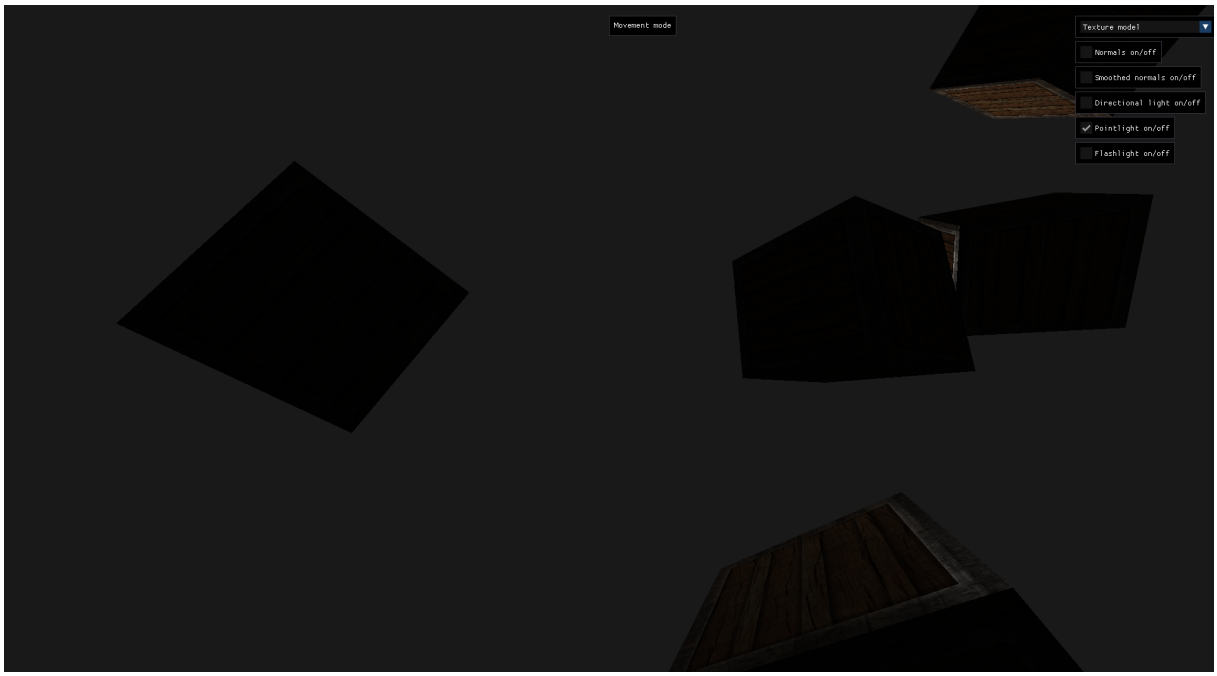


Рисунок 10: В это же время несглаженные нормали в области, где нет света работают без артефактов

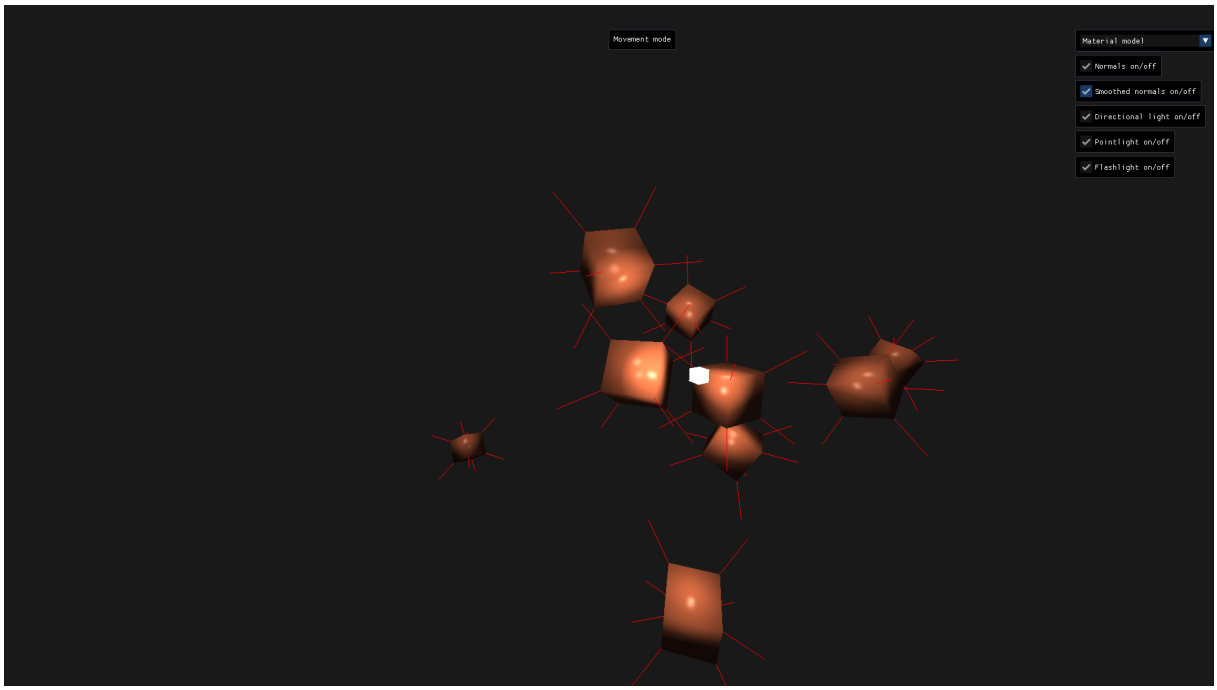


Рисунок 11: Пример работы освещения материала

Листинг

Object.cs

```
namespace _2lab.Objects;

using Shaders;
using BufferObjects;
using OpenTK.Graphics.OpenGL4;
using OpenTK.Mathematics;

public class Object : IObject
{
    private VertexBufferObject _vbo;
    private VertexArrayObject _vao;

    private float[] _vertices;

    private Shader _shader;

    private float[] _flashLightValues = {0.0f, 0.0f};
    private Vector3[] _pointLightValues =
    {
        new(0.05f, 0.05f, 0.05f),
        new(0.8f, 0.8f, 0.8f),
        new(1.0f, 1.0f, 1.0f)
    };
    private Vector3[] _dirLightValues =
    {
        new(0.05f, 0.05f, 0.05f),
        new(0.4f, 0.4f, 0.4f),
        new(0.5f, 0.5f, 0.5f)
```

```
};

private Vector3 _position;
private float _scale;

public Object(float[] vertices, Vector3 position, float scale)
{
    _vertices = vertices;
    _position = position;
    _scale = scale;

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

    var vertexLocation = _shader.GetAttribLocation("aPos");

    _vbo = new VertexBufferObject(_vertices);
    _vao = new VertexArrayObject(vertexLocation, 6);
    _vao.EnableArray(vertexLocation, 0);

    var normalLocation = _shader.GetAttribLocation("aNormal");

    _vao.EnableArray(normalLocation, 3 * sizeof(float));
}

public void Render(Camera camera, Vector3 lightPos, Vector3 position, float angle)
{
    _vao.Bind();

    _shader.Use();

    Matrix4 model = Matrix4.CreateTranslation(position);
    model *= Matrix4.CreateFromAxisAngle(new Vector3(1.0f, 0.3f, 0.5f), angle);
    _shader.SetMatrix4("model", model);
    _shader.SetMatrix4("view", camera.GetViewMatrix());
    _shader.SetMatrix4("projection", camera.GetProjectionMatrix());

    _shader.SetVector3("viewPos", camera.Position);

    _shader.SetVector3("material.ambient", new Vector3(1.0f, 0.5f, 0.31f));
    _shader.SetVector3("material.diffuse", new Vector3(1.0f, 0.5f, 0.31f));
    _shader.SetVector3("material.specular", new Vector3(0.5f, 0.5f, 0.5f));
    _shader.SetFloat("material.shininess", 32.0f);

    // Directional light
    _shader.SetVector3("dirLight.direction", new Vector3(-0.2f, -1.0f, -0.3f));
    _shader.SetVector3("dirLight.ambient", _dirLightValues[0]);
    _shader.SetVector3("dirLight.diffuse", _dirLightValues[1]);
    _shader.SetVector3("dirLight.specular", _dirLightValues[2]);

    // Point light
    _shader.SetVector3(\$"pointLights[0].position", lightPos);
    _shader.SetVector3(\$"pointLights[0].ambient", _pointLightValues[0]);
    _shader.SetVector3(\$"pointLights[0].diffuse", _pointLightValues[1]);
    _shader.SetVector3(\$"pointLights[0].specular", _pointLightValues[2]);
    _shader.SetFloat(\$"pointLights[0].constant", 1.0f);
```

```
        _shader.SetFloat(\$"pointLights[0].linear", 0.09f);
        _shader.SetFloat(\$"pointLights[0].quadratic", 0.032f);

        // Spot light
        _shader.SetVector3("spotLight.position", camera.Position);
        _shader.SetVector3("spotLight.direction", camera.Front);
        _shader.SetVector3("spotLight.ambient", new Vector3(0.0f, 0.0f, 0.0f));
        _shader.SetVector3("spotLight.diffuse", new Vector3(1.0f, 1.0f, 1.0f));
        _shader.SetVector3("spotLight.specular", new Vector3(1.0f, 1.0f, 1.0f));
        _shader.SetFloat("spotLight.constant", 1.0f);
        _shader.SetFloat("spotLight.linear", 0.09f);
        _shader.SetFloat("spotLight.quadratic", 0.032f);
                                                _shader.SetFloat("spotLight.cutOff",
MathF.Cos(MathHelper.DegreesToRadians(_flashLightValues[0])));
                                            _shader.SetFloat("spotLight.outerCutOff",
MathF.Cos(MathHelper.DegreesToRadians(_flashLightValues[1])));

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

    public void TurnOnFlashlight()
    {
        _flashLightValues[0] = 12.5f;
        _flashLightValues[1] = 17.5f;
    }

    public void TurnOffFlashlight()
    {
        _flashLightValues[0] = 0.0f;
        _flashLightValues[1] = 0.0f;
    }

    public void TurnOnPointlight()
    {
        _pointLightValues[0] = new Vector3(0.05f, 0.05f, 0.05f);
        _pointLightValues[1] = new Vector3(0.8f, 0.8f, 0.8f);
        _pointLightValues[2] = new Vector3(1.0f, 1.0f, 1.0f);
    }

    public void TurnOffPointlight()
    {
        _pointLightValues[0] = new Vector3(0.0f, 0.0f, 0.0f);
        _pointLightValues[1] = new Vector3(0.0f, 0.0f, 0.0f);
        _pointLightValues[2] = new Vector3(0.0f, 0.0f, 0.0f);
    }

    public void TurnOnDirlight()
    {
        _dirLightValues[0] = new Vector3(0.05f, 0.05f, 0.05f);
        _dirLightValues[1] = new Vector3(0.4f, 0.4f, 0.4f);
        _dirLightValues[2] = new Vector3(0.5f, 0.5f, 0.5f);
    }

    public void TurnOffDirlight()
    {
        _dirLightValues[0] = new Vector3(0.0f, 0.0f, 0.0f);
```

```
        _dirLightValues[1] = new Vector3(0.0f, 0.0f, 0.0f);
        _dirLightValues[2] = new Vector3(0.0f, 0.0f, 0.0f);
    }

    public void UpdateBuffers()
    {
        _vbo.Update(_vertices);
        _vao.Bind();
    }

    public void Dispose()
    {
        _vao.Dispose();
        _vbo.Dispose();

        _vertices = Array.Empty<float>();

        GC.SuppressFinalize(this);
    }
}
```

Texture.cs

```
namespace _2lab;

using OpenTK.Graphics.OpenGL4;
using StbImageSharp;

public class Texture
{
    public readonly int Handle;

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

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

        GL.ActiveTexture(TextureUnit.Texture0);
        GL.BindTexture(TextureTarget.Texture2D, handle);

        StbImage.stbi_set_flip_vertically_on_load(1);

                    ImageResult  image  =  ImageResult.FromStream(File.OpenRead(path),
ColorComponents.RedGreenBlueAlpha);

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

       GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter,
(int)TextureMinFilter.Linear);
       GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter,
(int)TextureMagFilter.Linear);
```

```
            GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapS,
(int)TextureWrapMode.Repeat);
            GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapT,
(int)TextureWrapMode.Repeat);

        GL.GenerateMipmap(GenerateMipmapTarget.Texture2D);

        return new Texture(handle);
    }

    public void Use(TextureUnit unit = TextureUnit.Texture0)
    {
        GL.ActiveTexture(unit);
        GL.BindTexture(TextureTarget.Texture2D, Handle);
    }
}
```

GUI.cs

```
namespace _2lab.GUI;

using ImGuiNET;
using OpenTK.Mathematics;

public class GUI
{
    private ImGuiController _controller;

    private Window _window;

    private readonly string[] _modelType;
    private readonly string[] _modeName;

    private int _selectedModelType;
    private bool _isClickedFlashlight;
    private bool _isClickedPointlight = true;
    private bool _isClickedDirectionallight = true;
    private bool _isClickedNormals;
    private bool _isClickedNormalsType;

    private int _appModes;

    private ImGuiWindowFlags _windowFlags =
        ImGuiWindowFlags.NoDecoration |
        ImGuiWindowFlags.AlwaysAutoResize |
        ImGuiWindowFlags.NoNav |
        ImGuiWindowFlags.NoSavedSettings |
        ImGuiWindowFlags.NoFocusOnAppearing |
        ImGuiWindowFlags.NoMove;

    public GUI(ImGuiController controller, Window window)
    {
        _controller = controller;
        _window = window;

        _modelType = new string[3]
```

```
        {
            "Material model",
            "Texture model",
            "Frame model"
        };

        _modeName = new string[2]
        {
            "Movement mode",
            "Cursor mode"
        };

          ImGui.PushStyleColor(ImGuiCol.Text, new System.Numerics.Vector4(1.0f, 1.0f,
1.0f, 1.0f));
        ImGui.PushStyleColor(ImGuiCol.FrameBg, new System.Numerics.Vector4(0.1f, 0.1f,
0.1f, 0.8f));
        ImGui.PushStyleColor(ImGuiCol.ChildBg, new System.Numerics.Vector4(0.0f, 0.0f,
0.0f, 1.0f));
       ImGui.PushStyleColor(ImGuiCol.WindowBg, new System.Numerics.Vector4(0.0f, 0.0f,
0.0f, 0.6f));
      ImGui.PushStyleColor(ImGuiCol.CheckMark, new System.Numerics.Vector4(1.0f, 1.0f,
1.0f, 0.6f));
    }

    public void Draw()
    {
        ImGui.SetNextWindowBgAlpha(1.0f);
        ImGui.SetNextWindowPos(new System.Numerics.Vector2(1700.0f, 40.0f));
        if (ImGui.Begin("View model", _windowFlags))
        {
           bool selectionChanged = ImGui.Combo("", ref _selectedModelType, _modelType,
_modelType.Length);

            if (selectionChanged)
            {
                switch (_selectedModelType)
                {
                    case 0:
                        _window.ChangeToMaterialModel();
                        break;

                    case 1:
                        _window.ChangeToTextureObject();
                        break;

                    case 2:
                        _window.ChangeToFrameObject();
                        break;
                }
            }

            ImGui.End();
        }

        if (_selectedModelType != 2)
        {
```

```
        ImGui.SetNextWindowBgAlpha(1.0f);
        ImGui.SetNextWindowPos(new System.Numerics.Vector2(1700.0f, 240.0f));
        if (ImGui.Begin("Flashlight", _windowFlags))
        {
            ImGui.Checkbox("Flashlight on/off", ref _isClickedFlashlight);
            if (_isClickedFlashlight)
            {
                _window.TurnOnFlashlight();
            }

            else
            {
                _window.TurnOffFlashlight();
            }
        }
    }

    if (_selectedModelType != 2)
    {
        ImGui.SetNextWindowBgAlpha(1.0f);
        ImGui.SetNextWindowPos(new System.Numerics.Vector2(1700.0f, 200.0f));
        if (ImGui.Begin("Pointlight", _windowFlags))
        {
            ImGui.Checkbox("Pointlight on/off", ref _isClickedPointlight);
            if (_isClickedPointlight)
            {
                _window.TurnOnPointLight();
            }

            else
            {
                _window.TurnOffPointLight();
            }
        }
    }

    if (_selectedModelType != 2)
    {
        ImGui.SetNextWindowBgAlpha(1.0f);
        ImGui.SetNextWindowPos(new System.Numerics.Vector2(1700.0f, 160.0f));
        if (ImGui.Begin("Directional light", _windowFlags))
        {
        ImGui.Checkbox("Directional light on/off", ref _isClickedDirectionallight);
            if (_isClickedDirectionallight)
            {
                _window.TurnOnDirLight();
            }

            else
            {
                _window.TurnOffDirLight();
            }
        }
    }

    ImGui.SetNextWindowBgAlpha(1.0f);
```

```
        ImGui.SetNextWindowPos(new System.Numerics.Vector2(1700.0f, 120.0f));
        if (ImGui.Begin("Smoothed normals", _windowFlags))
        {
            ImGui.Checkbox("Smoothed normals on/off", ref _isClickedNormalsType);
            if (_isClickedNormalsType)
            {
                _window.SmoothedNormals();
            }

            else
            {
                _window.UnSmoothedNormals();
            }
        }

        ImGui.SetNextWindowBgAlpha(1.0f);
        ImGui.SetNextWindowPos(new System.Numerics.Vector2(1700.0f, 80.0f));
        if (ImGui.Begin("Show normals", _windowFlags))
        {
            ImGui.Checkbox("Normals on/off", ref _isClickedNormals);
            if (_isClickedNormals)
            {
                _window.TurnOnNormals();
            }

            else
            {
                _window.TurnOffNormals();
            }
        }

        ImGui.SetNextWindowBgAlpha(1.0f);
        ImGui.SetNextWindowPos(new System.Numerics.Vector2(960.0f, 40.0f));
        ImGui.Begin("Text", _windowFlags);
        {
            ImGui.Text(_modeName[_window.CurrentAppMode]);
            ImGui.End();
        }
    }
}
```

Window.cs

```
namespace _2lab;

using Objects;
using GUI;

using OpenTK.Graphics.OpenGL4;
using OpenTK.Mathematics;
using OpenTK.Windowing.Common;
using OpenTK.Windowing.Desktop;
using OpenTK.Windowing.GraphicsLibraryFramework;

public class Window : GameWindow
{
```

```
    private readonly Vector3 _lightPos = new Vector3(1.2f, 1.0f, 2.0f);
    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 float _scale = 1.0f;

    private Object _object;
    private ObjectTexture _objectTexture;
    private ObjectFrame _objectFrame;
    private ObjectNormal _objectNormal;
    private Lamp _lamp;

    private Object _objectSmoothed;
    private ObjectTexture _objectTextureSmoothed;
    private ObjectNormal _objectNormalSmoothed;

    private IObject _currentObject;
    private IObject _currentObjectSmoothed;

    private Camera _camera;

    private Vector2 _lastPos;

    private GUI.GUI _gui;
    private ImGuiController _controller;

    private bool _canMove;
    private bool _smoothedNormals;
    private bool _firstMove = true;
    private bool _renderNormals;
    private bool _renderLamp = true;

    private enum AppMode
    {
        MovementMode,
        CursorMode
    }

    public int CurrentAppMode = (int)AppMode.CursorMode;

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

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

        GL.ClearColor(0.1f, 0.1f, 0.1f, 1.0f);

        GL.Enable(EnableCap.DepthTest);

        _object = new Object(_vertices, _cubePositions[0], _scale);
        _objectSmoothed = new Object(_verticesSmoothed, _cubePositions[0], _scale);

      _objectTexture = new ObjectTexture(_verticesTexture, _cubePositions[0], _scale);
              _objectTextureSmoothed  =  new  ObjectTexture(_verticesTextureSmoothed,
_cubePositions[0], _scale);

        _objectFrame = new ObjectFrame(_verticesFrame, _cubePositions[0], _scale);

        _objectNormal = new ObjectNormal(_normals, _cubePositions[0], _scale);
         _objectNormalSmoothed = new ObjectNormal(_normalsSmoothed, _cubePositions[0],
_scale);

        _lamp = new Lamp(_lightPos, _vertices);

        _currentObject = _object;
        _currentObjectSmoothed = _objectSmoothed;

        _controller = new ImGuiController(ClientSize.X, ClientSize.Y);
        _gui = new GUI.GUI(_controller, this);

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

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

        GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);
        for (int i = 0; i < _cubePositions.Length; i++)
        {
            float angle = 20.0f * i;

            if (_smoothedNormals)
            {
                  _currentObjectSmoothed.Render(_camera, _lightPos, _cubePositions[i],
angle);

                if (_renderNormals)
                {
                   _objectNormalSmoothed.Render(_camera, _lightPos, _cubePositions[i],
angle);
                }
            }

            else
            {
```

```
                _currentObject.Render(_camera, _lightPos, _cubePositions[i], angle);

                if (_renderNormals)
                {
                    _objectNormal.Render(_camera, _lightPos, _cubePositions[i], angle);
                }
            }
        }


        if (_renderLamp)
        {
            _lamp.Render(_camera);
        }

        _controller.Update(this, (float)e.Time);
        _gui.Draw();

        _controller.Render();

        SwapBuffers();
    }

    protected override void OnUpdateFrame(FrameEventArgs e)
    {
        base.OnUpdateFrame(e);
        if (!IsFocused)
        {
            return;
        }

        var input = KeyboardState;
        var mouse = MouseState;

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

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

        if (input.IsKeyDown(Keys.M))
        {
            _canMove = true;
            _lastPos = new Vector2(mouse.X, mouse.Y);

            CursorState = CursorState.Grabbed;
            CurrentAppMode = (int)AppMode.MovementMode;
        }

        if (input.IsKeyDown(Keys.N))
        {
            _canMove = false;

            CursorState = CursorState.Normal;
```

```
            CurrentAppMode = (int)AppMode.CursorMode;
        }

        if (input.IsKeyDown(Keys.P))
        {
            _camera.IsPerspective = true;
        }

        if (input.IsKeyDown(Keys.O))
        {
            _camera.IsPerspective = false;
        }

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

            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 OnResize(ResizeEventArgs e)
{
    base.OnResize(e);

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

public void ChangeToMaterialModel()
{
    _currentObject = _object;
    _currentObjectSmoothed = _objectSmoothed;
}

public void ChangeToTextureObject()
{
    _currentObject = _objectTexture;
    _currentObjectSmoothed = _objectTextureSmoothed;
}

public void ChangeToFrameObject()
{
    _currentObject = _objectFrame;
    _currentObjectSmoothed = _objectFrame;
}

public void TurnOnFlashlight()
{
    if (_currentObject != _objectFrame)
    {
        _currentObject.TurnOnFlashlight();
        _currentObjectSmoothed.TurnOnFlashlight();
    }
}

public void TurnOffFlashlight()
{
    if (_currentObject != _objectFrame)
    {
        _currentObject.TurnOffFlashlight();
        _currentObjectSmoothed.TurnOffFlashlight();
    }
}

public void TurnOnPointLight()
{
    if (_currentObject != _objectFrame)
    {
        _currentObject.TurnOnPointlight();
        _currentObjectSmoothed.TurnOnPointlight();
        _renderLamp = true;
```

```
    }
}

public void TurnOffPointLight()
{
    if (_currentObject != _objectFrame)
    {
        _currentObject.TurnOffPointlight();
        _currentObjectSmoothed.TurnOffPointlight();
        _renderLamp = false;
    }
}

public void TurnOnDirLight()
{
    if (_currentObject != _objectFrame)
    {
        _currentObject.TurnOnDirlight();
        _currentObjectSmoothed.TurnOnDirlight();
    }
}

public void TurnOffDirLight()
{
    if (_currentObject != _objectFrame)
    {
        _currentObject.TurnOffDirlight();
        _currentObjectSmoothed.TurnOffDirlight();
    }
}

public void TurnOnNormals()
{
    _renderNormals = true;
}

public void TurnOffNormals()
{
    _renderNormals = false;
}

public void SmoothedNormals()
{
    _smoothedNormals = true;
}

public void UnSmoothedNormals()
{
    _smoothedNormals = false;
}

private readonly float[] _normals =
{
    -0.5f, -0.5f, -0.5f, -0.5f, -0.5f, -1.0f, // Front face
     0.5f, -0.5f, -0.5f, 0.5f, -0.5f, -1.0f,
     0.5f,  0.5f, -0.5f, 0.5f,  0.5f, -1.0f,
```

```
    -0.5f,  0.5f, -0.5f, -0.5f,  0.5f, -1.0f,

    -0.5f, -0.5f,  0.5f, -0.5f,  -0.5f,  1.0f, // Back face
     0.5f, -0.5f,  0.5f, 0.5f,  -0.5f,  1.0f,
     0.5f,  0.5f,  0.5f, 0.5f,  0.5f,  1.0f,
    -0.5f,  0.5f,  0.5f, -0.5f,  0.5f,  1.0f,

    -0.5f,  0.5f,  0.5f, -1.0f,  0.5f,  0.5f, // Left face
    -0.5f,  0.5f, -0.5f, -1.0f,  0.5f,  -0.5f,
    -0.5f, -0.5f, -0.5f, -1.0f,  -0.5f,  -0.5f,
    -0.5f, -0.5f,  0.5f, -1.0f, -0.5f,  0.5f,

     0.5f,  0.5f,  0.5f,  1.0f, 0.5f,  0.5f,  // Right face
     0.5f,  0.5f, -0.5f,  1.0f, 0.5f, -0.5f,
     0.5f, -0.5f, -0.5f,  1.0f, -0.5f, -0.5f,
     0.5f, -0.5f,  0.5f,  1.0f, -0.5f,  0.5f,

    -0.5f, -0.5f, -0.5f, -0.5f, -1.0f, -0.5f, // Bottom face
     0.5f, -0.5f, -0.5f,  0.5f, -1.0f, -0.5f,
     0.5f, -0.5f,  0.5f,  0.5f, -1.0f,  0.5f,
    -0.5f, -0.5f,  0.5f, -0.5f, -1.0f,  0.5f,

    -0.5f,  0.5f, -0.5f, -0.5f,  1.0f, -0.5f, // Top face
     0.5f,  0.5f, -0.5f,  0.5f,  1.0f, -0.5f,
     0.5f,  0.5f,  0.5f,  0.5f,  1.0f,  0.5f,
    -0.5f,  0.5f,  0.5f, -0.5f,  1.0f,  0.5f,
};

private readonly float[] _vertices =
{
     // Position          Normal
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f, // Front face
     0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,

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

    -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f, // Back face
     0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,

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

    -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f, // Left face
    -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,

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

     0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f, // Right face
```

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

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

    -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f, // Bottom face
     0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
     0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,

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

    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f, // Top face
     0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,

     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f
};

private readonly float[] _verticesTexture =
{
    // Positions          Normals              Texture coords
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  0.0f, 0.0f,
     0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  1.0f, 0.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  1.0f, 1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  1.0f, 1.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  0.0f, 1.0f,
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  0.0f, 0.0f,

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

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

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

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

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

    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,  0.0f, 1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,  1.0f, 1.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,  1.0f, 0.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,  1.0f, 0.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,  0.0f, 0.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,  0.0f, 1.0f
};

private readonly float[] _verticesFrame =
{
     // Position          Normal
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f, // Front face
     0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,

    -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f, // Back face
     0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
     0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
    -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,

    -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f, // Left face
    -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,

     0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f, // Right face
     0.5f,  0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
     0.5f,  0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
     0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
     0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
     0.5f, -0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
     0.5f, -0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,

    -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f, // Bottom face
     0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
     0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
```

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

    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f, // Top face
     0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f
};

private readonly float[] _normalsSmoothed =
{
    -0.5f, -0.5f, -0.5f, -1.0f, -1.0f, -1.0f,
     0.5f, -0.5f, -0.5f, 1.0f, -1.0f, -1.0f,
    -0.5f,  0.5f, -0.5f, -1.0f, 1.0f, -1.0f,
     0.5f,  0.5f, -0.5f, 1.0f,  1.0f, -1.0f,

    -0.5f, -0.5f, 0.5f, -1.0f, -1.0f, 1.0f,
     0.5f, -0.5f, 0.5f, 1.0f, -1.0f, 1.0f,
    -0.5f,  0.5f, 0.5f, -1.0f, 1.0f, 1.0f,
     0.5f,  0.5f, 0.5f, 1.0f,  1.0f, 1.0f,
};

private readonly float[] _verticesSmoothed =
{
     // Position          Normal
    -0.5f, -0.5f, -0.5f,  -1.0f,  -1.0f, -1.0f, // Front face
     0.5f, -0.5f, -0.5f,  1.0f,  -1.0f, -1.0f,
     0.5f,  0.5f, -0.5f,  1.0f,  1.0f, -1.0f,

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

    -0.5f, -0.5f,  0.5f,  -1.0f,  -1.0f,  1.0f, // Back face
     0.5f, -0.5f,  0.5f,  1.0f,  -1.0f,  1.0f,
     0.5f,  0.5f,  0.5f,  1.0f,  1.0f,  1.0f,

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

    -0.5f,  0.5f,  0.5f, -1.0f,  1.0f,  1.0f, // Left face
    -0.5f,  0.5f, -0.5f, -1.0f,  1.0f,  -1.0f,
    -0.5f, -0.5f, -0.5f, -1.0f,  -1.0f,  -1.0f,

    -0.5f, -0.5f, -0.5f, -1.0f,  -1.0f,  -1.0f,
    -0.5f, -0.5f,  0.5f, -1.0f,  -1.0f,  1.0f,
    -0.5f,  0.5f,  0.5f, -1.0f,  1.0f,  1.0f,
```

```
     0.5f,  0.5f,  0.5f,  1.0f,  1.0f,  1.0f, // Right face
     0.5f,  0.5f, -0.5f,  1.0f,  1.0f,  -1.0f,
     0.5f, -0.5f, -0.5f,  1.0f,  -1.0f,  -1.0f,

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

    -0.5f, -0.5f, -0.5f,  -1.0f, -1.0f,  -1.0f, // Bottom face
     0.5f, -0.5f, -0.5f,  1.0f, -1.0f,  -1.0f,
     0.5f, -0.5f,  0.5f,  1.0f, -1.0f,  1.0f,

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

    -0.5f,  0.5f, -0.5f,  -1.0f,  1.0f,  -1.0f, // Top face
     0.5f,  0.5f, -0.5f,  1.0f,  1.0f,  -1.0f,
     0.5f,  0.5f,  0.5f,  1.0f,  1.0f,  1.0f,

     0.5f,  0.5f,  0.5f,  1.0f,  1.0f,  1.0f,
    -0.5f,  0.5f,  0.5f,  -1.0f,  1.0f,  1.0f,
    -0.5f,  0.5f, -0.5f,  -1.0f,  1.0f,  -1.0f
};

private readonly float[] _verticesTextureSmoothed =
{
    // Positions          Normals              Texture coords
    -0.5f, -0.5f, -0.5f,  -1.0f,  -1.0f, -1.0f,  0.0f, 0.0f,
     0.5f, -0.5f, -0.5f,  1.0f,  -1.0f, -1.0f,  1.0f, 0.0f,
     0.5f,  0.5f, -0.5f,  1.0f,  1.0f, -1.0f,  1.0f, 1.0f,
     0.5f,  0.5f, -0.5f,  1.0f,  1.0f, -1.0f,  1.0f, 1.0f,
    -0.5f,  0.5f, -0.5f,  -1.0f,  1.0f, -1.0f,  0.0f, 1.0f,
    -0.5f, -0.5f, -0.5f,  -1.0f,  -1.0f, -1.0f,  0.0f, 0.0f,

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

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

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

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

        -0.5f,  0.5f, -0.5f,  -1.0f,  1.0f,  -1.0f,  0.0f, 1.0f,
         0.5f,  0.5f, -0.5f,  1.0f,  1.0f,  -1.0f,  1.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  1.0f,  1.0f,  1.0f,  1.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  1.0f,  1.0f,  1.0f,  1.0f, 0.0f,
        -0.5f,  0.5f,  0.5f,  -1.0f,  1.0f,  1.0f,  0.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,  -1.0f,  1.0f,  -1.0f,  0.0f, 1.0f
    };
}
```