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 №4
по дисциплине «Компьютерная графика»

ИНТЕРАКТИВНОЕ СОЗДАНИЕ КРИВЫХ И ПОВЕРХНОСТЕЙ С ИСПОЛЬЗОВАНИЕМ СПЛАЙНОВ

Группа: ПМ-02 ДЗЮБЛО ПАВЕЛ
ДАНЧЕНКО ИВАН

Преподаватели: ЗАДОРОЖНЫЙ А.Г.

Новосибирск, 2023

Цель работы

Реализовать программу, отображающую график функции, получаемой в результате использования соответствующего сплайна.

Средства реализации приложения

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

Возможности приложения

- Есть возможность переключаться между тремя состояниями:
  - **Spectate** - просмотр полученного изображения
  - **Workspace** - все слои, кроме выбранного, прозрачные
  - **Edit** - можно добавлять или убирать точки для выбранного слоя

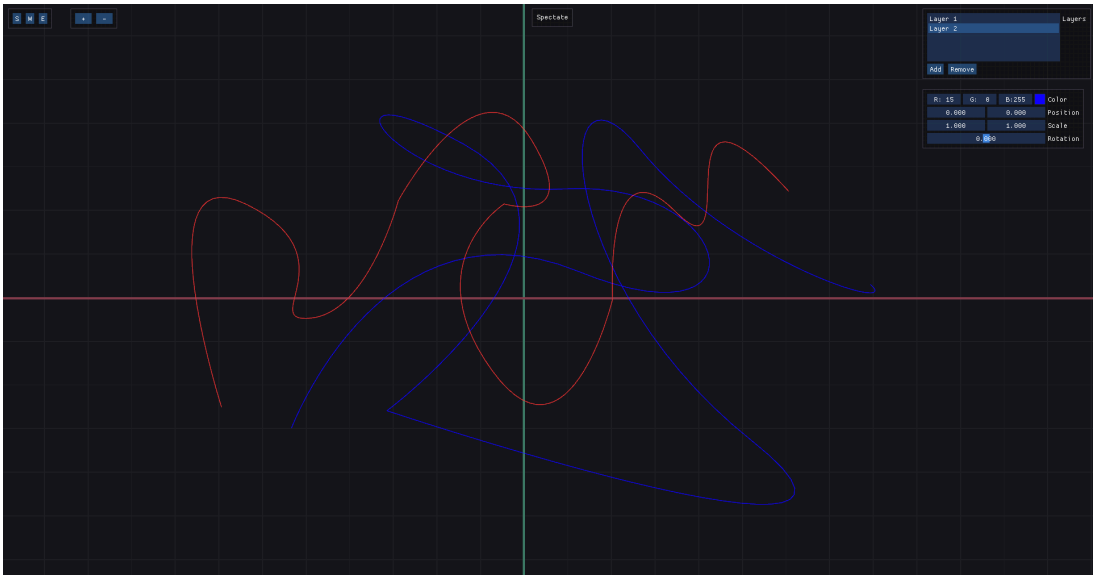
Текущий режим можно узнать по надписи сверху.

В режиме **Workspace** происходит основная работа с выбранным слоем, а то есть:

1. Перемещение - изменяя параметр **Position** или с помощью перехода в режим *Grab*(клавиша G) с последующим изменением положения с помощью мыши.
2. Масштабирование - информация дублируется с первого пунка, только изменяется параметр **Scale** или переход в режим *Scale*(S).
3. Поворот - параметр **Rotation** или переход в режим *Rotate*(R).
4. Изменение цвета - изменение с помощью аттрибута **Color**.
5. Добавить новый слой - нажатие на кнопку **Add** или с помощью бинда на клавишу N.
6. Удалить выбранный слой - нажатие на кнопку **Remove**

Также можно двигать камеру с помощью зажатого колёсика мыши и масштабировать с помощью прокрутки.

Пример работы приложения



Листинг

```
using Silk.NET.OpenGL;
using Silk.NET.Windowing;
using Silk.NET.Input;
using Silk.NET.Maths;
using System.Numerics;
using Lab4.App.States;
using Lab4.Core;

namespace Lab4.App
{
    public class App
    {
        private IWindow _window;
        private AppContext _context = new AppContext();

        public List<Layer> Layers = new List<Layer>();
        private Camera _camera = new Camera();
        private Origin _origin;
        private CellField _cellField;

                        private   System.Numerics.Vector2   previousPosition   =   new
System.Numerics.Vector2(0.0f, 0.0f);

        public bool Dragging { get; set; } = false;
        public bool BezierDragging { get; set; } = false;
        public int LayerID { get; set; } = 0;
        public int TotalLayers { get; private set; } = 0;

     private Dictionary<string, AppState> _states = new Dictionary<string, AppState>();
        public string CurrentState { get; private set; } = "Spectate";
        public Vector2 MouseOffset { get; private set; } = new Vector2(0.0f, 0.0f);
        public Vector2 MousePosition { get => _context.MousePosition; }

        public float CameraScale { get => _camera.Transform.Scale.X; }

        public App()
        {
            var options = WindowOptions.Default;

            options.Title = "Simple Drawer";
            options.Size = new Vector2D<int>(1280, 720);
            options.Samples = 4;

            _window = Window.Create(options);

            _window.Load += OnWindowLoad;
            _window.Resize += OnWindowResize;
            _window.Render += OnWindowRender;
            _window.Closing += OnWindowClosing;

            _states.Add("Spectate", new States.Spectate(this));
            _states.Add("Workspace", new States.Workspace(this));
            _states.Add("Edit", new States.Edit(this));
            _states.Add("Grab", new States.Grab(this));
            _states.Add("Scale", new States.Scale(this));
```

```
    _states.Add("Rotate", new States.Rotate(this));

    _window.Run();
}

public void Close()
{
    _window.Close();
}

public void MouseVisible(bool visibility)
{
    if (visibility)
    {
        _context.Mouse!.Cursor.CursorMode = CursorMode.Normal;
    }
    else
    {
        _context.Mouse!.Cursor.CursorMode = CursorMode.Disabled;
    }

}

public void NextLayer()
{
    Layers[LayerID].Transperent = 0.2f;
    LayerID = (LayerID == Layers.Count - 1) ? 0 : LayerID + 1;
    Layers[LayerID].Transperent = 1.0f;
}

public void PreviousLayer()
{
    Layers[LayerID].Transperent = 0.2f;
    LayerID = (LayerID == 0) ? Layers.Count - 1 : LayerID - 1;
    Layers[LayerID].Transperent = 1.0f;
}

public void UpdateLayerGuiData(int layerID)
{
    if (_context.Gui!.CurrentLayer != layerID && CurrentState != "Spectate")
    {
        MakeAllLayersTransperent(0.2f);
        Layers[layerID].Transperent = 1.0f;
    }

    Layers[LayerID].DrawElementInfo = false;
    Layers[layerID].DrawElementInfo = true;

    _context.Gui!.CurrentLayer = layerID;

    var position = Layers[layerID].Transform.Position;
    var scale = Layers[layerID].Transform.Scale;
    var rotation = Layers[layerID].Transform.Rotation;

    _context.Gui!.Color = Layers[layerID].Color;
    _context.Gui!.Position = new Vector2(position.X, position.Y);
```

```
            _context.Gui!.Scale = new Vector2(scale.X, scale.Y);

            var angle = (float)(System.Math.Asin(rotation.Z) * 2.0);

            _context.Gui!.Rotation = angle * 180.0f / (float)System.Math.PI;
            _context.Gui!.LayerNames = Layers.Select(layer => layer.Name).ToArray();
        }

        public void UpdateLayerDataWithGui(int layerID)
        {

            var position = _context.Gui!.Position;
            var scale = _context.Gui!.Scale;

            float rotation = (float)(_context.Gui!.Rotation * System.Math.PI / 180.0);
            var quaternion = new Quaternion();

            quaternion.W = (float)System.Math.Cos(rotation / 2.0);
            quaternion.Z = (float)System.Math.Sin(rotation / 2.0);

            Layers[layerID].Color = _context.Gui!.Color;
             Layers[layerID].Transform.Position = new Vector3(position.X, position.Y,
Layers[layerID].Transform.Position.Z);
                    Layers[layerID].Transform.Scale = new Vector3(scale.X, scale.Y,
Layers[layerID].Transform.Scale.Z);
            Layers[layerID].Transform.Rotation = quaternion;
        }

        public void AddLayer()
        {
            TotalLayers += 1;
               Layers.Add(new Layer(_context.Gl!, Color.FromHSV(0.0f, 0.77f, 0.95f),
$"Layer {TotalLayers}", LayerID * 0.001f));

            UpdateLayerGuiData(LayerID);
        }

        public void RemoveLayer()
        {
            if (Layers.Count > 1)
            {
                Layers.RemoveAt(LayerID);
                LayerID = Layers.Count == LayerID ? LayerID - 1 : LayerID;
            }
            else
            {
                Layers[0].Clear();
            }

            UpdateLayerGuiData(LayerID);
        }

        public void AddHoverPoint()
        {
            Layers[LayerID].DrawLast = false;
            Layers[LayerID].AddPoint(HoverablePoint());
```

```
        }

        public void UpdateHoverPoint()
        {
            if (Layers[LayerID].DrawLast == false)
            {
                Layers[LayerID].ChangeLastPoint(HoverablePoint());
            }
        }

        public void UpdateHoverPoint(Vertex position, Vertex offset)
        {
            if (Layers[LayerID].DrawLast == false)
            {
                Layers[LayerID].ChangeLastPoint(position, offset);
            }
        }

        public void RemoveHoverPoint()
        {
            Layers[LayerID].DrawLast = true;
            Layers[LayerID].RemoveLastPoint();
        }

        public void AddPoint(Vertex position, Vertex offset)
        {
            Layers[LayerID].AddPoint(position, offset);
        }

        public void AddPointByMousePosition()
        {
            Layers[LayerID].AddPoint(HoverablePoint());
        }

        public Vertex HoverablePoint()
        {
            return new Vertex(
                    (-_camera.CameraPosition.X + (2.0f * (_context.MousePosition.X) /
_window.Size.X - 1.0f)) / (_camera.Transform.Scale.X),
                      (-_camera.CameraPosition.Y * _camera.ViewportRatioXY - (2.0f *
(_context.MousePosition.Y) / _window.Size.Y - 1.0f)) / (_camera.Transform.Scale.X *
_camera.ViewportRatioXY)
            );
        }

        private void OnWindowLoad()
        {
            _context.AttachWindow(_window);

            if (_context.Mouse != null)
            {
                _context.Mouse.MouseDown += (IMouse mouse, MouseButton button) =>
                {
                    Dragging = button == MouseButton.Middle ? true : Dragging;
                };
```

```
                _context.Mouse.MouseUp += (IMouse mouse, MouseButton button) =>
                {
                    Dragging = button == MouseButton.Middle ? false : Dragging;
                };

                   _context.Mouse.MouseMove += (IMouse mouse, System.Numerics.Vector2
position) =>
                {
                    Vector2 offset = position - previousPosition;
                    offset.Y *= -1.0f;
                    previousPosition = position;

                    MouseOffset = offset;

                    if (Dragging)
                    {
                        Vector3 pos = _camera.CameraPosition;

                        pos.X += MouseOffset.X * 0.002f;
                        pos.Y += MouseOffset.Y * 0.002f;

                          _camera.CameraPosition = new System.Numerics.Vector3(pos.X,
pos.Y, pos.Z);
                    };
                };

                _context.Mouse.Scroll += (IMouse _mouse, ScrollWheel scroll) =>
                {
                    // Looks like crap
                    var scale = _camera.Transform.Scale;

                    scale.X += scroll.Y * 0.1f;
                    scale.X = scale.X <= 2.5f ? scale.X : 2.5f;
                    scale.X = scale.X >= 0.1f ? scale.X : 0.1f;

                    scale.Y = scale.X;
                    _camera.Transform.Scale = scale;
                };

                _context.Gui!.OnCameraZoomForawardButtonPressed += () =>
                {
                    var scale = _camera.Transform.Scale;

                    scale.X += 0.1f;
                    scale.X = scale.X <= 2.5f ? scale.X : 2.5f;
                    scale.X = scale.X >= 0.1f ? scale.X : 0.1f;

                    scale.Y = scale.X;
                    _camera.Transform.Scale = scale;
                };

                _context.Gui!.OnCameraZoomBackButtonPressed += () =>
                {
                    var scale = _camera.Transform.Scale;

                    scale.X -= 0.1f;
```

```
            scale.X = scale.X <= 2.5f ? scale.X : 2.5f;
            scale.X = scale.X >= 0.1f ? scale.X : 0.1f;

            scale.Y = scale.X;
            _camera.Transform.Scale = scale;
        };

    }

    AddLayer();
    UpdateLayerGuiData(LayerID);

    _origin = new Origin(_context.Gl!);
    _cellField = new CellField(_context.Gl!);

    _context.Gui!.OnAddLayerButtonPressed += AddLayer;
    _context.Gui!.OnRemoveLayerButtonPressed += RemoveLayer;

    _context.Gui!.OnSpectateModeButtonPressed += () =>
    {
        // TODO: Bug:
        // Pressed button affects on field as mouse click
        // So we remove the point with RemoveHoverVertex();
        if (CurrentState == "Edit")
        {
            RemoveHoverPoint();
        }

        ChangeState("Spectate");
    };
    _context.Gui!.OnWorkspaceModeButtonPressed += () =>
    {
        // TODO: Bug:
        // Pressed button affects on field as mouse click
        // So we remove the point with RemoveHoverVertex();
        if (CurrentState == "Edit")
        {
            RemoveHoverPoint();
        }

        ChangeState("Workspace");
    };
    _context.Gui!.OnEditModeButtonPressed += () => ChangeState("Edit");

    Layers[0].Transperent = 0.2f;

    ChangeState("Spectate");
}

private void OnWindowRender(double delta)
{
    Color background = Color.FromHSV(0.7f, 0.2f, 0.1f);
    _context.FrameSetup(background);

    _cellField.Draw(_camera, Layers[_context.Gui!.CurrentLayer].Transform);
```

```
    foreach (var layer in Layers)
    {
        layer.Draw(_camera);
    }

    _context.Gui!.Process((float)delta);
    _states[CurrentState].Render(delta);


    if (_context.Gui!.CurrentLayer != LayerID)
    {
        if (CurrentState != "Spectate")
        {
            MakeAllLayersTransperent(0.2f);
            Layers[_context.Gui!.CurrentLayer].Transperent = 1.0f;
        }

        UpdateLayerGuiData(_context.Gui!.CurrentLayer);
    }

    if (CurrentState != "Spectate")
    {
        _origin.Draw(_camera, Layers[_context.Gui!.CurrentLayer].Transform);
    }


    LayerID = _context.Gui!.CurrentLayer;
    UpdateLayerDataWithGui(LayerID);
}

private void OnWindowResize(Vector2D<int> size)
{
    _context.Gl!.Viewport(size);
    _context.Gui!.SetViewportSize(new Vector2(size.X, size.Y));
    _camera.ViewportSize = new Vector2(size.X, size.Y);
}

private void OnWindowClosing()
{
    _window.Load -= OnWindowLoad;
    _window.Resize -= OnWindowResize;
    _window.Render -= OnWindowRender;
    _window.Closing -= OnWindowClosing;
}

internal void MakeAllLayersTransperent(float alpha)
{
    foreach (var layer in Layers)
    {
        layer.Transperent = alpha;
    }
}

internal void ChangeState(string newStateName)
{
    if (!_states.ContainsKey(newStateName))
```

```
            {
                return;
            }

            _states[CurrentState].Exit();

            DisposeStateEventHangling(CurrentState);
            ListenStateEventHangling(newStateName);

            _context.Gui!.ModeName = newStateName;
            CurrentState = newStateName;

            _states[CurrentState].Enter();
        }

        private void ListenStateEventHangling(string stateName)
        {
            if (_context.Keyboard != null)
            {
                _context.Keyboard.KeyUp += _states[stateName].OnKeyUp;
                _context.Keyboard.KeyDown += _states[stateName].OnKeyDown;
            }

            if (_context.Mouse != null)
            {
                _context.Mouse.MouseUp += _states[stateName].OnMouseUp;
                _context.Mouse.MouseDown += _states[stateName].OnMouseDown;
                _context.Mouse.MouseMove += _states[stateName].OnMouseMove;
                _context.Mouse.Scroll += _states[stateName].OnMouseScroll;
            }
        }

        private void DisposeStateEventHangling(string stateName)
        {
            if (_context.Keyboard != null)
            {
                _context.Keyboard.KeyUp -= _states[stateName].OnKeyUp;
                _context.Keyboard.KeyDown -= _states[stateName].OnKeyDown;
            }

            if (_context.Mouse != null)
            {
                _context.Mouse.MouseUp -= _states[stateName].OnMouseUp;
                _context.Mouse.MouseDown -= _states[stateName].OnMouseDown;
                _context.Mouse.MouseMove -= _states[stateName].OnMouseMove;
                _context.Mouse.Scroll -= _states[stateName].OnMouseScroll;
            }
        }
    }
}
```

```
using Silk.NET.OpenGL;
using Lab4.Math;

namespace Lab4.Core
{
    public record struct QubicBezierCurveElement
```

```
{
    public QubicBezierCurveElement(Vertex center)
    {
        Center = center;
        Left = new Vertex(center.X, center.Y);
        Right = new Vertex(center.X, center.Y);
    }

    public QubicBezierCurveElement(Vertex center, Vertex left, Vertex right)
    {
        Center = center;
        Left = left;
        Right = right;
    }

    public Vertex Center { get; set; }
    public Vertex Left { get; set; }
    public Vertex Right { get; set; }
}

public class Layer
{
    public Transform Transform = new Transform();
    private List<Vertex> _vertices;
    private Canvas _canvas;
    private List<QubicBezierCurveElement> _curve;
    private Color _color;
    private float _zIndex = 0.0f;

    public string Name { get; set; } = "Layer";

    public Layer(GL context, Color color, string name, float zIndex)
    {
        _color = color;
        Name = name;
        _vertices = new List<Vertex>();
        _curve = new List<QubicBezierCurveElement>();
        _canvas = new Canvas(context, color);
    }

    public void AddPoint(Vertex vertex)
    {
        _curve.Add(new QubicBezierCurveElement(vertex));

        _canvas.AttachData(GetCurvePointsBuffer());
        _canvas.AttachLinesData(GetLinesBuffer());
    }

    public void AddPoint(Vertex vertex, Vertex offset)
    {
        _curve.Add(new QubicBezierCurveElement(
            vertex,
            new Vertex(vertex.X - offset.X, vertex.Y - offset.Y),
            new Vertex(vertex.X + offset.X, vertex.Y + offset.Y)
        ));
```

```
    _canvas.AttachData(GetCurvePointsBuffer());
    _canvas.AttachLinesData(GetLinesBuffer());
}

public void RemoveLastPoint()
{
    if (_curve.Count > 1)
    {
        _curve.RemoveAt(_curve.Count - 1);

        _canvas.AttachData(GetCurvePointsBuffer());
        _canvas.AttachLinesData(GetLinesBuffer());
    }
}

private float[] GetLinesBuffer()
{
    float[] data = new float[_curve.Count * 2 * 3];

    if (_curve.Count > 1)
    {
        data[0] = _curve[0].Center.X;
        data[1] = _curve[0].Center.Y;
        data[2] = _zIndex;

        data[3] = _curve[0].Right.X;
        data[4] = _curve[0].Right.Y;
        data[5] = _zIndex;
    }

    if (_curve.Count > 2)
    {
        data[data.Length - 6] = _curve[_curve.Count - 1].Left.X;
        data[data.Length - 5] = _curve[_curve.Count - 1].Left.Y;
        data[data.Length - 4] = _zIndex;

        data[data.Length - 3] = _curve[_curve.Count - 1].Center.X;
        data[data.Length - 2] = _curve[_curve.Count - 1].Center.Y;
        data[data.Length - 1] = _zIndex;
    }

    for (int iElem = 1, iData = 6; iElem < _curve.Count - 1; iElem++)
    {
        data[iData++] = _curve[iElem].Left.X;
        data[iData++] = _curve[iElem].Left.Y;
        data[iData++] = _zIndex;

        data[iData++] = _curve[iElem].Right.X;
        data[iData++] = _curve[iElem].Right.Y;
        data[iData++] = _zIndex;
    }

    return data;
}

private float[] GetCurvePointsBuffer()
```

```
{
    float[] data = new float[(_curve.Count - 1) * 4 * 3 + 9];

    if (_curve.Count == 1)
    {
        data = new float[9];

        data[0] = _curve[_curve.Count - 1].Left.X;
        data[1] = _curve[_curve.Count - 1].Left.Y;
        data[2] = _zIndex;

        data[3] = _curve[_curve.Count - 1].Center.X;
        data[4] = _curve[_curve.Count - 1].Center.Y;
        data[5] = _zIndex;
    }
    else
    {
        data = new float[(_curve.Count - 1) * 4 * 3 + 3];

        for (int iElem = 0, iData = 0; iElem < _curve.Count - 1; iElem++)
        {
            data[iData++] = _curve[iElem].Center.X;
            data[iData++] = _curve[iElem].Center.Y;
            data[iData++] = _zIndex;

            data[iData++] = _curve[iElem].Right.X;
            data[iData++] = _curve[iElem].Right.Y;
            data[iData++] = _zIndex;

            data[iData++] = _curve[iElem + 1].Left.X;
            data[iData++] = _curve[iElem + 1].Left.Y;
            data[iData++] = _zIndex;

            data[iData++] = _curve[iElem + 1].Center.X;
            data[iData++] = _curve[iElem + 1].Center.Y;
            data[iData++] = _zIndex;
        }
    }


    data[data.Length - 3] = _curve[_curve.Count - 1].Right.X;
    data[data.Length - 2] = _curve[_curve.Count - 1].Right.Y;
    data[data.Length - 1] = _zIndex;

    return data;
}

public void ChangeLastPoint(Vertex vertex)
{
    if (_curve.Count != 0)
    {
        Vertex to = _curve.Last().Right.To(vertex);

        var element = new QubicBezierCurveElement(vertex);
        _curve[_curve.Count - 1] = element;
```

```
            // _canvas.ChangeLastPoint(vertex.X, vertex.Y, 0.0f);
            // TODO: wow, needs to be optimized
            float[] data = GetCurvePointsBuffer();
            _canvas.AttachData(data);
        }
    }

    public void ChangeLastPoint(Vertex center, Vertex offset)
    {
        if (_curve.Count != 0)
        {
            var element = new QubicBezierCurveElement(
                center,
                new Vertex(center.X - offset.X, center.Y - offset.Y),
                new Vertex(center.X + offset.X, center.Y + offset.Y)
            );

            _curve[_curve.Count - 1] = element;

            // _canvas.ChangeLastPoint(vertex.X, vertex.Y, 0.0f);
            // TODO: wow, needs to be optimized
            float[] data = GetCurvePointsBuffer();
            _canvas.AttachData(data);
        }
    }

    public int GetCurveElementsCount() => _curve.Count;

    public Color Color
    {
        get => _color;
        set
        {
            _color = value;
            _canvas.Color = _color;
        }
    }

    public float Transperent
    {
        get => _color.Alpha;
        set
        {
            _color.Alpha = value;
            _canvas.Color = _color;
        }
    }

    public void Draw(Camera camera)
    {
        _canvas.Draw(camera, Transform);
    }

    public void Clear()
    {
        Transform = new Transform();
```

```
            _color = new Color();
            _curve.Clear();

            float[] data = new float[0] { };
            _canvas.AttachData(data);
            _canvas.AttachLinesData(data);
        }

        public bool DrawLast
        {
            set => _canvas.DrawLast = value;
            get => _canvas.DrawLast;
        }

        public bool ApplyLayerTransform
        {
            set => _canvas.ApplyCanvasTransform = value;
            get => _canvas.ApplyCanvasTransform;
        }

        public bool DrawElementInfo
        {
            set => _canvas.DrawElementInfo = value;
            get => _canvas.DrawElementInfo;
        }
    }
}
```

```
using Silk.NET.OpenGL;
using Silk.NET.Windowing;
using Silk.NET.Input;
using Silk.NET.OpenGL.Extensions.ImGui;
using System.Numerics;
using Lab4.Core;

namespace Lab4.App
{
    public class Gui
    {
        private ImGuiNET.ImGuiWindowFlags _windowFlags =
            ImGuiNET.ImGuiWindowFlags.NoDecoration |
            ImGuiNET.ImGuiWindowFlags.AlwaysAutoResize |
            ImGuiNET.ImGuiWindowFlags.NoNav |
            ImGuiNET.ImGuiWindowFlags.NoSavedSettings |
            ImGuiNET.ImGuiWindowFlags.NoFocusOnAppearing |
            ImGuiNET.ImGuiWindowFlags.NoMove;
        private ImGuiController _controller;
        private ImGuiNET.ImGuiViewport _viewport;
        private Vector3 _color = Vector3.One;
        private Vector2 _position = Vector2.Zero;
        private Vector2 _scale = Vector2.One;
        private float _rotation = 0.0f;
        private int _currentLayer = 0;
        private string[] _layers = new string[0];

        private GL _gl;
        private IView _window;
```

```
private IInputContext _input;

public string ModeName = "default";

public Color Color
{
    get => new Color(_color.X, _color.Y, _color.Z);
    set
    {
        _color.X = value.Red;
        _color.Y = value.Green;
        _color.Z = value.Blue;
    }
}

public Vector2 Position { get => _position; set => _position = value; }
public Vector2 Scale { get => _scale; set => _scale = value; }
public float Rotation { get => _rotation; set => _rotation = value; }
public int CurrentLayer { get => _currentLayer; set => _currentLayer = value; }
public string[] LayerNames { get => _layers; set => _layers = value; }


public delegate void AddLayerButtonPressed();
public delegate void RemoveLayerButtonPressed();
public delegate void SpectateModeButtonPressed();
public delegate void WorkspaceModeButtonPressed();
public delegate void EditModeButtonPressed();
public delegate void CameraZoomForawardButtonPressed();
public delegate void CameraZoomBackButtonPressed();

public AddLayerButtonPressed OnAddLayerButtonPressed;
public RemoveLayerButtonPressed OnRemoveLayerButtonPressed;
public SpectateModeButtonPressed OnSpectateModeButtonPressed;
public WorkspaceModeButtonPressed OnWorkspaceModeButtonPressed;
public EditModeButtonPressed OnEditModeButtonPressed;
public CameraZoomForawardButtonPressed OnCameraZoomForawardButtonPressed;
public CameraZoomBackButtonPressed OnCameraZoomBackButtonPressed;

public Gui(GL gl, IView window, IInputContext input)
{
    _gl = gl;
    _window = window;
    _input = input;

    _controller = new ImGuiController(gl, window, input);
    _viewport = new ImGuiNET.ImGuiViewport();
}

public Gui(GL gl, IView window, IInputContext input, ImGuiFontConfig config)
{
    _gl = gl;
    _window = window;
    _input = input;

    _controller = new ImGuiController(_gl, _window, _input, config);
    _viewport = new ImGuiNET.ImGuiViewport();
```

```
        }

        public void SetViewportSize(Vector2 size)
        {
            _viewport.Size = size;
        }

        public void Process(float delta)
        {
            _controller.Update(delta);

            ProcessMode();
            ProcessModeInfo();
            ProcessCameraButtons();
            ProcessLayerSelector();
            ProcessLayerProperties();

            // ImGuiNET.ImGui.ShowDemoWindow();

            _controller.Render();
        }

        private void ProcessCameraButtons()
        {
            ImGuiNET.ImGui.SetNextWindowBgAlpha(0.35f);
            ImGuiNET.ImGui.SetNextWindowPos(new Vector2(120.0f, 10.0f));

            ImGuiNET.ImGui.Begin("Camera", _windowFlags);
            {
                ImGuiNET.ImGui.BeginGroup();
                {
                       bool cameraZoomForward = ImGuiNET.ImGui.Button(" + ") ? true :
false;
                    ImGuiNET.ImGui.SameLine();
                    bool cameraZoomBack = ImGuiNET.ImGui.Button(" - ") ? true : false;

                   if (cameraZoomForward && OnCameraZoomForawardButtonPressed != null)
                    {
                        OnCameraZoomForawardButtonPressed();
                    }

                    if (cameraZoomBack && OnCameraZoomBackButtonPressed != null)
                    {
                        OnCameraZoomBackButtonPressed();
                    }
                }
                ImGuiNET.ImGui.EndGroup();
            }
        }

        private void ProcessLayerSelector()
        {
            ImGuiNET.ImGui.SetNextWindowBgAlpha(0.35f);
            ImGuiNET.ImGui.SetNextWindowPos(
                new Vector2((_viewport.Size.X - 300.0f),
                10.0f
```

```
            ));

            ImGuiNET.ImGui.Begin("Layers", _windowFlags);
            {
                ImGuiNET.ImGui.PushItemWidth(234.0f);

                    ImGuiNET.ImGui.BeginListBox("Layers", new Vector2(0.0f, 5.0f *
ImGuiNET.ImGui.GetTextLineHeightWithSpacing()));
                {
                    for (int idx = 0; idx < _layers.Length; idx++)
                    {
                        bool isSelected = (CurrentLayer == idx);

                        if (ImGuiNET.ImGui.Selectable(_layers[idx], isSelected))
                        {
                            CurrentLayer = idx;
                        }

                        if (isSelected)
                        {
                            ImGuiNET.ImGui.SetItemDefaultFocus();
                        }
                    }
                }
                ImGuiNET.ImGui.EndListBox();

                ImGuiNET.ImGui.BeginGroup();
                {
                    bool addClicked = ImGuiNET.ImGui.Button("Add") ? true : false;
                    ImGuiNET.ImGui.SameLine();
                  bool removeClicked = ImGuiNET.ImGui.Button("Remove") ? true : false;

                    if (addClicked && OnAddLayerButtonPressed != null)
                    {
                        OnAddLayerButtonPressed();
                    }

                    if (removeClicked && OnRemoveLayerButtonPressed != null)
                    {
                        OnRemoveLayerButtonPressed();
                    }
                }
                ImGuiNET.ImGui.EndGroup();
            }
            ImGuiNET.ImGui.End();
        }

        private void ProcessMode()
        {
            ImGuiNET.ImGui.SetNextWindowBgAlpha(0.35f);
            ImGuiNET.ImGui.SetNextWindowPos(new Vector2(10.0f, 10.0f));

            ImGuiNET.ImGui.Begin("App Modes", _windowFlags);
            {
                ImGuiNET.ImGui.BeginGroup();
                {
```

```
                    bool spectateClicked = ImGuiNET.ImGui.Button("S") ? true : false;
                    ImGuiNET.ImGui.SameLine();
                    bool workspaceClicked = ImGuiNET.ImGui.Button("W") ? true : false;
                    ImGuiNET.ImGui.SameLine();
                    bool editClicked = ImGuiNET.ImGui.Button("E") ? true : false;

                    if (spectateClicked && OnSpectateModeButtonPressed != null)
                    {
                        OnSpectateModeButtonPressed();
                    }

                    if (workspaceClicked && OnWorkspaceModeButtonPressed != null)
                    {
                        OnWorkspaceModeButtonPressed();
                    }

                    if (editClicked && OnEditModeButtonPressed != null)
                    {
                        OnEditModeButtonPressed();
                    }
                }
                ImGuiNET.ImGui.EndGroup();
            }
            ImGuiNET.ImGui.End();
        }

        private void ProcessLayerProperties()
        {
            ImGuiNET.ImGui.SetNextWindowBgAlpha(0.35f);
            ImGuiNET.ImGui.SetNextWindowPos(
                new Vector2((_viewport.Size.X - 300.0f),
                152.0f
            ));

            ImGuiNET.ImGui.Begin("Layer Properties", _windowFlags);
            {
                ImGuiNET.ImGui.ColorEdit3("Color", ref _color);

                ImGuiNET.ImGui.DragFloat2("Position", ref _position, 0.01f);
                ImGuiNET.ImGui.DragFloat2("Scale", ref _scale, 0.01f);
               ImGuiNET.ImGui.SliderFloat("Rotation", ref _rotation, -180.0f, 180.0f);
            }
            ImGuiNET.ImGui.End();
        }

        private void ProcessModeInfo()
        {
            ImGuiNET.ImGui.SetNextWindowBgAlpha(0.35f);
            ImGuiNET.ImGui.SetNextWindowPos(
              new Vector2((_viewport.Size.X - ImGuiNET.ImGui.CalcTextSize(ModeName).X)
* 0.5f,
                10.0f
            ));

            ImGuiNET.ImGui.Begin("Text", _windowFlags);
            {
```

```
                ImGuiNET.ImGui.Text(ModeName);
            }
            ImGuiNET.ImGui.End();
        }

        public void SetFont(string source, int fontSize)
        {
            ImGuiFontConfig config = new ImGuiFontConfig(source, fontSize);

            _controller = new ImGuiController(_gl, _window, _input, config);
        }
    }
}
```

```
using System.Numerics;
using Lab4.Math;
using ReMath = Lab4.Math;

namespace Lab4.Core
{
    public class Camera
    {
        public Transform Transform;
        private Vector3 _cameraPosition = new Vector3(0.0f, 0.0f, 1.0f);
        private Vector3 _cameraFront = new Vector3(0.0f, 0.0f, -1.0f);
        private Vector3 _cameraUp = Vector3.UnitY;
        private Vector3 _cameraDirection = Vector3.Zero;

        public Camera()
        {
            Transform = new Transform();
            Transform.Position = _cameraPosition;
        }

        public Vector3 CameraPosition
        {
            get => _cameraPosition;
            set
            {
                _cameraPosition = value;
                Transform.Position = _cameraPosition;
            }
        }

        // TODO: Bullshit. I need Viewport class, that uses Camera
        public Vector2 ViewportSize { get; set; }
        public float ViewportRatioXY { get => ViewportSize.X / ViewportSize.Y; }
        public float ViewportRatioYX { get => ViewportSize.Y / ViewportSize.X; }
    }
}
```