Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления

Кафедра Интеллектуальных информационных технологий

**ОТЧЁТ**

по дисциплине «Графический интерфейс интеллектуальных систем»

Лабораторная работа №7

Тема: Триангуляция. Построение диаграммы Вороного

Выполнили: Демидовец Д.В.

Козырев Д.А.

гр. 221703

Проверил: Сальников Д. А.

Минск 2025

**Задание к лабораторной работе**

Разработать графическую программу, выполняющую триангуляцию Делоне и построение диаграммы Вороного по заданному набору точек.

**Ход работы**

1. Изучение теоретических основ алгоритмов.
2. Реализация алгоритмов построения диаграмм в программной среде.
3. Проверка отображения в редакторе.

**Описание алгоритмов**

Триангуляция Делоне — это способ соединения точек треугольниками так, чтобы ни одна точка не лежала внутри окружности, описанной около любого треугольника. Он максимизирует минимальные углы треугольников (избегает "длинных и тонких" треугольников).

Диаграмма Вороного — это разбиение плоскости на области вокруг заданных точек так, что каждая область содержит все точки, которые ближе к своей "центральной" точке, чем к любой другой.

**Листинг кода**

function VoronoiDelaunay() {

*const* [points, setPoints] = useState([]);

*const* [mode, setMode] = useState("delaunay");

*const* [status, setStatus] = useState("");

*const* [inputCoords, setInputCoords] = useState({ x: "", y: "" });

*const* plotRef = useRef(null);

*const* handleAddPoint = () => {

*const* x = parseFloat(inputCoords.x);

*const* y = parseFloat(inputCoords.y);

    if (isNaN(x) || isNaN(y) || x < 0 || x > 10 || y < 0 || y > 10) {

      setStatus("Ошибка: координаты от 0 до 10");

*return*;

    }

    setPoints(prev => [**...**prev, [x, y]]);

    setInputCoords({ x: "", y: "" });

    setStatus(`Добавлена точка (${x.toFixed(1)}, ${y.toFixed(1)})`);

  };

*const* generateDelaunay = () => {

    if (points.length < 3) {

      setStatus("Нужно минимум 3 точки!");

*return*;

    }

    setStatus("Триангуляция Делоне построена");

  };

*const* generateVoronoi = () => {

    if (points.length < 3) {

      setStatus("Нужно минимум 3 точки!");

*return*;

    }

    setStatus("Диаграмма Вороного построена");

  };

**Результаты работы**

Диаграмма Делоне:

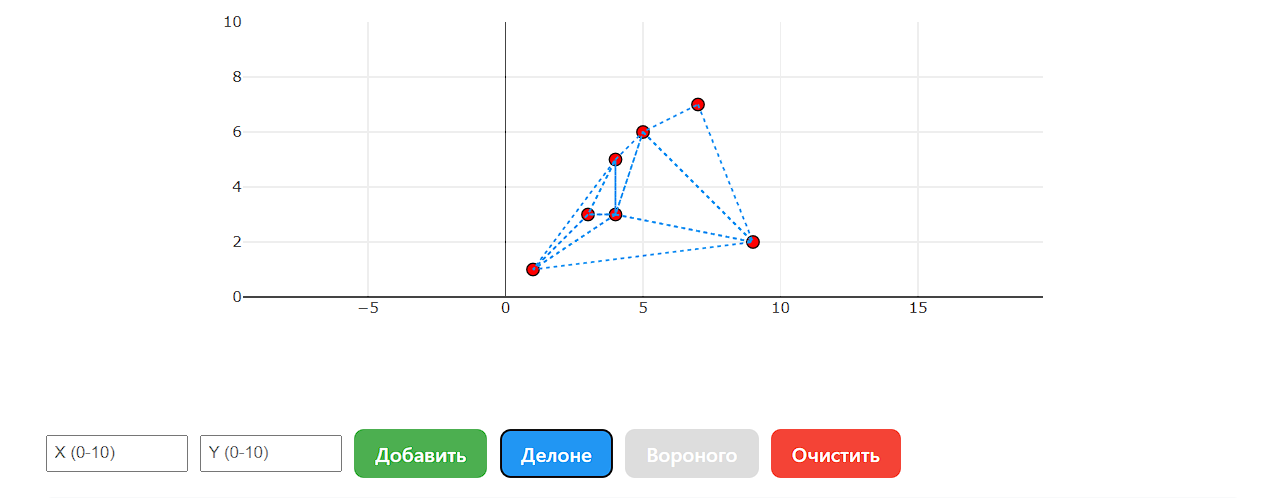
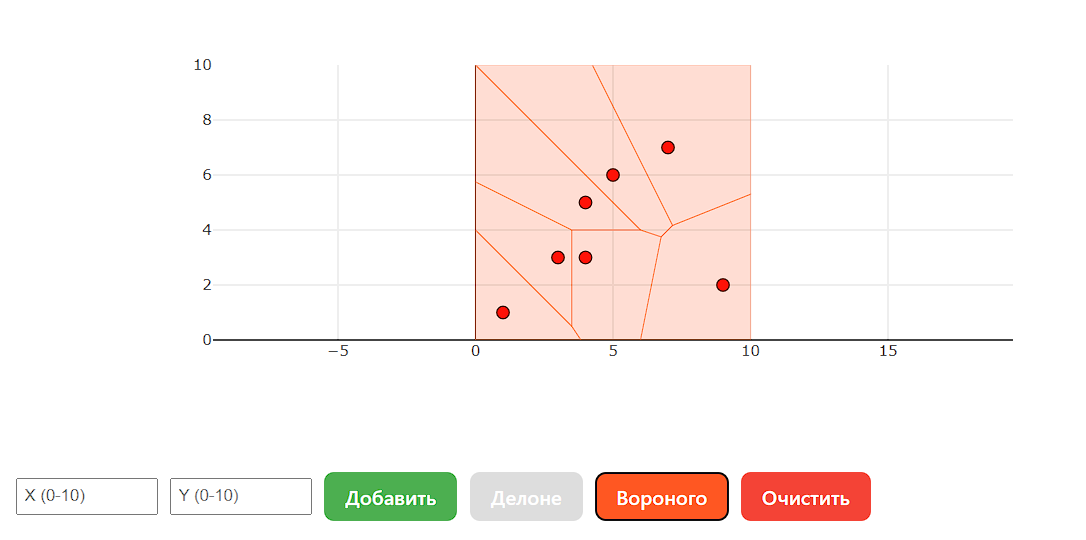


Диаграмма Вороного:



**Вывод**

В ходе лабораторной работы были изучены и реализованы основные алгоритмы построения диаграммы Вороного и триангуляции Делоне для заданного множества точек на плоскости. Полученные результаты демонстрируют важность данных методов в задачах визуализации, моделирования, интерполяции и пространственного анализа. Практическое применение таких диаграмм охватывает широкий спектр областей — от компьютерной графики и ГИС до машинного обучения и робототехники.

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления

Кафедра Интеллектуальных информационных технологий

**ОТЧЁТ**

по дисциплине «Графический интерфейс интеллектуальных систем»

Лабораторная работа №8

Тема: Удаление невидимых линий и поверхностей

Выполнили: Демидовец Д.В.

Козырев Д.А.

гр. 221703

Проверил: Сальников Д. А.

Минск 2025

**Задание к лабораторной работе**

Разработать графическую программу, выполняющую отсечение невидимых линий двухмерных объектов и удаление невидимых граней трехмерных объектов.

**Ход работы**

1. Изучение теоретических основ алгоритмов.
2. Реализация алгоритмов отсечения и удаления в программной среде.
3. Проверка отображения в редакторе.

**Описание алгоритмов**

Алгоритм Коэна-Сазерленда используется для отсечения отрезков по прямоугольному окну в 2D. Основные шаги алгоритма:

1. Определение кода положения точки (Outcode):
   1. Каждой точке присваивается 4-битный код, отражающий её положение относительно окна отсечения (внутри/слева/справа/снизу/сверху).
   2. Функция computeCode(x, y) возвращает этот код.
2. Проверка отрезка:
   1. Для каждой пары точек (ребра полигона), вычисляются их коды.
   2. Если обе точки внутри окна (их коды равны 0), отрезок целиком принимается.
   3. Если логическое И (AND) кодов не ноль — обе точки находятся по одну сторону за пределами окна, и отрезок отклоняется.
   4. Иначе — отрезок пересекает окно, и его часть внутри окна находится с помощью пересечения с одной из сторон окна.
3. Итеративное отсечение:
   1. Отсекаемую точку заменяют на точку пересечения с границей окна.
   2. Код обновляется, и процесс повторяется, пока не будет решено принять или отклонить отрезок.

**Удаление невидимых граней в трехмерном пространстве** реализуется **встроенными средствами библиотеки Three.js.** Это поведение происходит автоматически и опирается на стандартный алгоритм, называемый **Z-буферизацией (z-buffering):**

1. Для каждой вершины вычисляется её проекция на экран (с учетом перспективной матрицы).
2. Для каждого фрагмента (пикселя) объекта считается глубина (Z-координата в системе камеры).
3. Эта глубина сравнивается с текущим значением в Z-буфере.
   1. Если новая глубина меньше (т.е. ближе к камере), то фрагмент рендерится, и Z-буфер обновляется.
   2. Иначе — фрагмент игнорируется (невидимая грань не отображается).

**Листинг кода**

// 2D

*const* cohenSutherlandClip = () => {

    if (hullPoints.length < 3) *return*;

*const* INSIDE = 0, LEFT = 1, RIGHT = 2, BOTTOM = 4, TOP = 8;

*const* clipWindow = { x1: 2, y1: 2, x2: 8, y2: 8 };

*const* computeCode = (x, y) => {

*let* code = INSIDE;

      if (x < clipWindow.x1) code |= LEFT;

      else if (x > clipWindow.x2) code |= RIGHT;

      if (y < clipWindow.y1) code |= BOTTOM;

      else if (y > clipWindow.y2) code |= TOP;

*return* code;

    };

// 3D

*const* Transform3D = () => {

*// Инициализация рендерера*

    renderer.current = new THREE.WebGLRenderer({

      antialias: true,

      alpha: true

    });

    renderer.current.setSize(600, 400);

    renderer.current.setClearColor(0xffffff, 0);

    containerRef.current.appendChild(renderer.current.domElement);

*// Настройка камеры для вида "3 грани"*

    camera.current.position.set(5, 5, 5);

    camera.current.lookAt(0, 0, 0);

*// Освещение*

    scene.current.add(new THREE.AmbientLight(0xffffff, 0.8));

*const* directionalLight = new THREE.DirectionalLight(0xffffff, 0.5);

    directionalLight.position.set(-5, 5, 5);

    scene.current.add(directionalLight);

*// Создание куба*

*const* geometry = new THREE.BoxGeometry(3, 3, 3);

*// Основной материал (белый)*

*const* material = new THREE.MeshPhongMaterial({

      color: 0xffffff,

      polygonOffset: true,

      polygonOffsetFactor: 1,

      polygonOffsetUnits: 1

    });

*// Геометрия для видимых ребер*

*const* edges = new THREE.EdgesGeometry(geometry, 30);

*const* edgeMaterial = new THREE.LineBasicMaterial({

      color: 0x000000,

      linewidth: 2

    });

*const* cube = new THREE.Mesh(geometry, material);

*const* wireframe = new THREE.LineSegments(edges, edgeMaterial);

*const* group = new THREE.Group();

    group.add(cube);

    group.add(wireframe);

    scene.current.add(group);

*// OrbitControls*

    controls.current = new OrbitControls(camera.current, renderer.current.domElement);

    controls.current.enableDamping = true;

    controls.current.dampingFactor = 0.05;

*// Анимация*

*const* animate = () => {

      requestAnimationFrame(animate);

      controls.current.update();

      renderer.current.render(scene.current, camera.current);

    };

    animate();

*return* () => {

      containerRef.current?.removeChild(renderer.current.domElement);

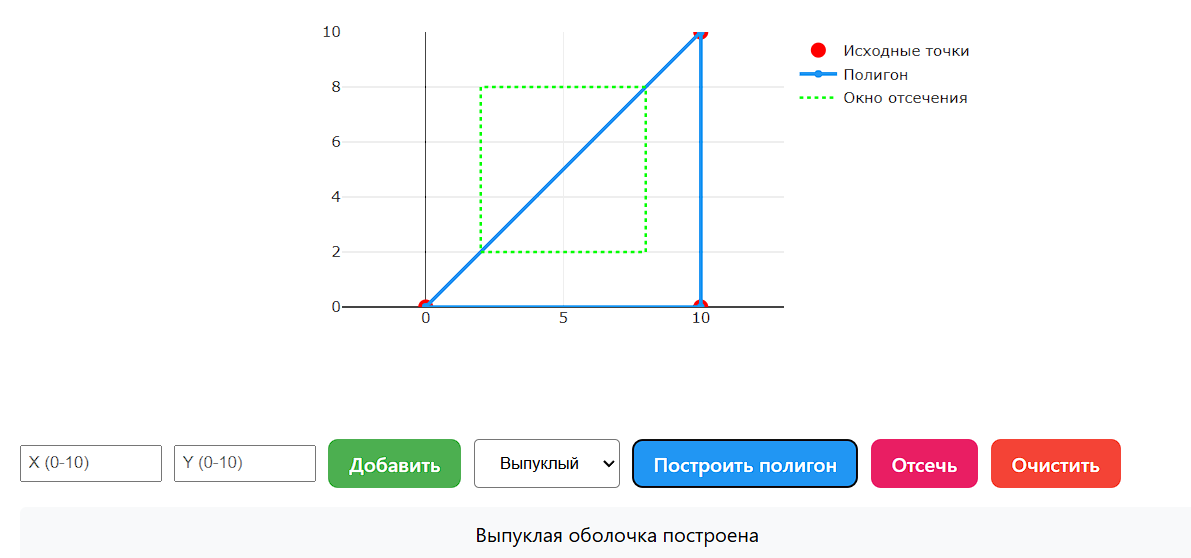
      controls.current?.dispose();

    };

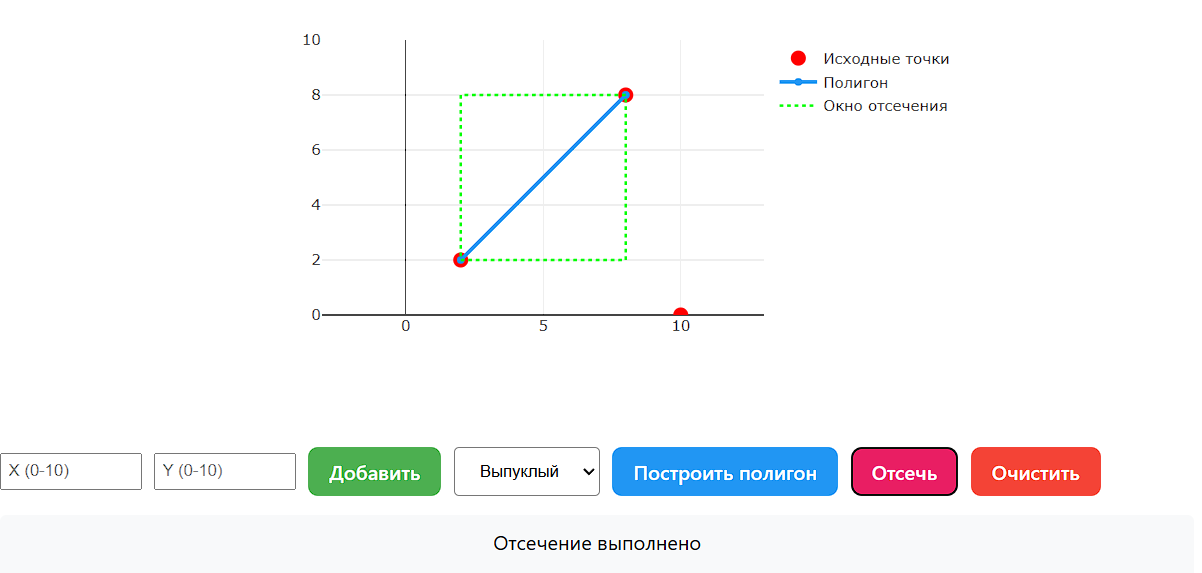
  }, []);

**Результаты работы**

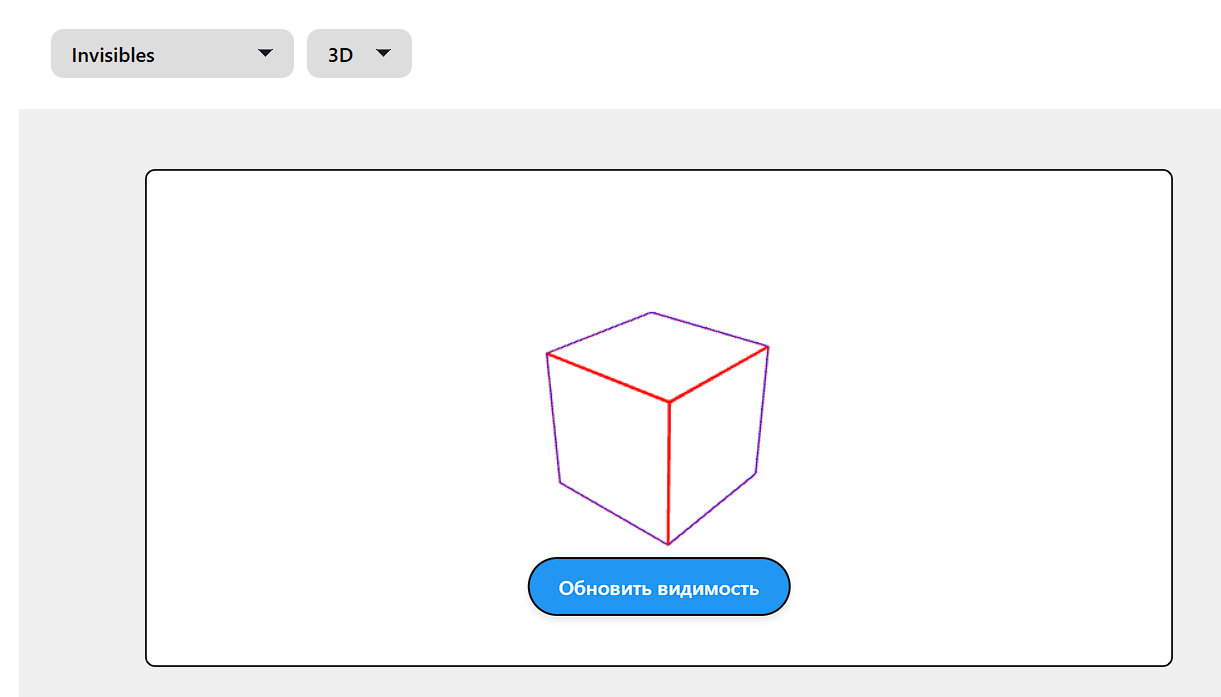
Изначальный полигон:



Отсечение в двухмерном пространстве:



Удаление невидимых граней в трехмерном пространстве:



**Вывод**

В ходе лабораторной работы были реализованы алгоритмы двумерного отсечения и визуализации трехмерных объектов с удалением невидимых граней. Работа способствовала усвоению принципов графического рендеринга, включая методы отсечения в 2D-пространстве и автоматическое удаление невидимых поверхностей в 3D-сценах, что имеет важное значение для оптимизации и визуализации в компьютерной графике.