Geometric Algorithms
Assignment 3

Laura Baakman (s1869140)

September 30, 2014

A

Point in Triangle

We try to determine if the point $\mathbf{Q}$ lies in the triangle defined by the points $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_3}$. To this end we define the vectors $\mathbf{v}_1 = \mathbf{P_2} - \mathbf{P_1}$ and $\mathbf{v_2} = \mathbf{P_3} - \mathbf{P_1}$, see Figure 1. Each point $\mathbf{P}$ inside the grey area in this figure can be described as:

$$\mathbf{P} = \mathbf{P_1} + a \cdot \mathbf{v_1} + b \cdot \mathbf{v_2}. \tag{1}$$

For all points to the right of $\mathbf{P_1}$ $a > 0$ and $b > 0$. The points that define the triangle can all be expressed according to (1):

$$\mathbf{P_1} = \mathbf{P_1} + 0 \cdot \mathbf{v_1} + 0 \cdot \mathbf{v_2} \tag{2}$$
$$\mathbf{P_2} = \mathbf{P_1} + 1 \cdot \mathbf{v_1} + 0 \cdot \mathbf{v_2} \tag{3}$$
$$\mathbf{P_3} = \mathbf{P_1} + 0 \cdot \mathbf{v_1} + 1 \cdot \mathbf{v_2}. \tag{4}$$

If $a + b == 1$ the point lies on the edge between $\mathbf{P_2}$ and $\mathbf{P_3}$. Based on Equation 2 through 4 we find that a point $\mathbf{Q}$ lies on the triangle if it can be expressed according to (1) with $a, b \in (0, 1)$ and with $a + b < 1$.

Solving the resulting equation with Mathematica, see Listing 1, gives us expressions for a and b, namely:

$$a = -\frac{-\mathbf{P_{1,0}P_{3,1}} + \mathbf{P_{1,0}Q1} + \mathbf{P_{1,1}P_{3,0}} - \mathbf{P_{1,1}Q_0} - \mathbf{P_{3,0}} \cdot \mathbf{Q_1} + \mathbf{P_{3,1}} \cdot \mathbf{Q_0}}{-\mathbf{P_{1,0}P_{2,1}} + \mathbf{P_{1,0}} \cdot \mathbf{P_{3,1}} + \mathbf{P_{1,1}} \cdot \mathbf{P_{20}} - \mathbf{P_{1,1}P_{3,0}} - \mathbf{P_{2,0}} \cdot \mathbf{P_{3,1}} + \mathbf{P_{2,1}} \cdot \mathbf{P_{3,0}}} \tag{5}$$

$$b = -\frac{-\mathbf{P_{1,0}} \cdot \mathbf{P_{2,1}} + \mathbf{P_{1,0}} \cdot \mathbf{Q_1} + \mathbf{P_{1,1}} \cdot \mathbf{P_{2,0}} - \mathbf{P_{1,1}Q_0} - \mathbf{P_{2,0}} \cdot \mathbf{Q_1} + \mathbf{P_{2,1}} \cdot \mathbf{Q_0}}{\mathbf{P_{1,0}P_{2,1}} - \mathbf{P_{1,0}} \cdot \mathbf{P_{3,1}} - \mathbf{P_{1,1}} \cdot \mathbf{P_{2,0}} + \mathbf{P_{1,1}P_{3,0}} + \mathbf{P_{2,0}} \cdot \mathbf{P_{3,1}} - \mathbf{P_{2,1}} \cdot \mathbf{P_{3,0}}} \tag{6}$$

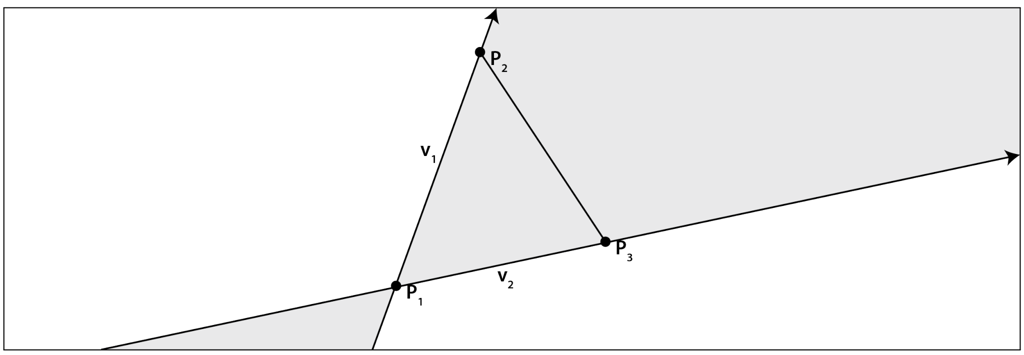

Figure 1: A triangle defined by the points $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_3}$, with the vectors $\mathbf{v_1}$ and $\mathbf{v_2}$. The grey area covers all points that can be described according to (1).

Listing 1: Mathematica code used to compute the to compute a and b.

```
p1 = {p10, p11};
p2 = {p20, p21};
p3 = {p30, p31};
v1 = p2 - p1;
v2 = p3 - p1;

p4 = p1 + a * v1 + b * v2;
p41 = Part[p4, 1] == Q0;
p42 = Part[p4, 2] == Q1;
solution = Solve[{p41, p42}, {a, b}]
```

Listing 2: The method `point_in_triangle()` in the module `triangle`.

```
def point_in_triangle(triangle, point):
    """
    Return true if the point lies in the triangle.

    Input:
        triangle: List of three points, where each point is a list
            with the x and y coordinate of an vertex of the triangle.
        point: List with the x and y coordinate of the point.
    """
    [p1, p2, p3] = triangle
    v1_cross_v2 = (
        -p1[1] * p2[0] + p1[0] * p2[1] + p1[1] * p3[0]
        - p2[1] * p3[0] - p1[0] * p3[1] + p2[0] * p3[1]
    )
    if(v1_cross_v2):
        a_numerator = (
            p1[1] * p3[0] - p1[0] * p3[1] - p1[1] * point[0] +
            p3[1] * point[0] + p1[0] * point[1] - p3[0] * point[1]
        )
        a = a_numerator / v1_cross_v2
        if(a > 0 and a < 1):
            b_numerator = (
                p1[1] * p2[0] - p1[0] * p2[1] - p1[1] * point[0] +
                p2[1] * point[0] + p1[0] * point[1] - p2[0] * point[1]
            )
            b = - b_numerator / v1_cross_v2
            return (b > 0 and b < 1) and (a + b < 1)
    return False
```

where $\mathbf{P_{r,s}}$ represents the s'th element of the point r and $\mathbf{Q_t}$ represent the t'th element of the point $\mathbf{Q}$.

The denominator of (5) and (6) are the same, this is the magnitude of $\mathbf{v_1} \times \mathbf{v_2}$ where the vectors are made three-dimensional by adding a z-coordinate of zero, If that cross product is zero $\mathbf{v_1}$ and $\mathbf{v_2}$ are parallel and (1) can thus be only used to represent points on a line parallel to $\mathbf{v_1}$ and through $\mathbf{P_1}$.

The method presented above is implemented in the method `point_in_triangle()` in the module `triangle`, see Listing 2.

Finding the Triangle Containing the Point

We have simply checked for all triangles that result from the triangulation if the point lies inside that triangle, see Listing 3 for the method `find_containing_triangle()`. Figure 2 shows the triangulation of 100 points generated randomly with the seed set to 505. The triangle that contains the point `lp` (305, 350) is shown in green, as is the point it self. The containing triangle is defined by the points: (4.35e2, 2.75e2), (2.93e2, 3.71e2), (3.35e2, 2.46e2).

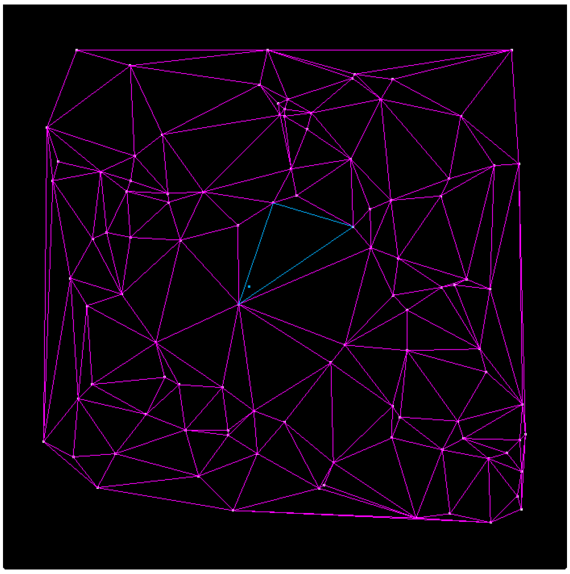

Figure 2: The triangulation of one hundred points, the triangle shown in green contains the point `lp`, also shown in green.

Listing 3: The method `find_containing_triangle()`.

```
def find_containing_triangle(point):
    """
    Find the triangle of the triangulation that contains the point point.

    The triangle is returned as an index in the global array triPts.

    """
    global triPts, lp, xl, yl
    for triangle in triPts:
        [p1, p2, p3] = triangle
        if(point_in_triangle(
            [
                [xl[p1], yl[p1]],
                [xl[p2], yl[p2]],
                [xl[p3], yl[p3]]
            ],
            lp
        )):
            print (
                "The triangle that contains the point ({lp_x},{lp_y}):"
                "({p1_x}, {p1_y}), ({p2_x}, {p2_y}), ({p3_x}, {p3_y})"
                .format(
                    lp_x=lp[0], lp_y=lp[1],
                    p1_x=xl[p1], p1_y=yl[p1],
                    p2_x=xl[p2], p2_y=yl[p2],
                    p3_x=xl[p3], p3_y=yl[p3],
                )
            )
            return triangle
```

Listing 4: Mathematica code used to solve Equation 9.

```
eq1 = lam1 p1x + (1 - lam1) p2x == lam2  p3x + (1 - lam2) p4x
eq2 = lam1 p1y + (1 - lam1) p2y == lam2  p3y + (1 - lam2) p4y
Solve[eq1 == eq2 {lam1, lam2}]
```

B

Line Segment Intersection

To find the intersection of the following two line segments

$$s_1 = \lambda_1 \cdot \mathbf{P_1} + (1 - \lambda_1) \cdot \mathbf{P_2} \qquad \text{for } 0 \leq \lambda_1 \leq 1 \tag{7}$$
$$s_2 = \lambda_2 \cdot \mathbf{P_3} + (1 - \lambda_2) \cdot \mathbf{P_4} \qquad \text{for } 0 \leq \lambda_2 \leq 1 \tag{8}$$

we need to solve the equation:

$$\begin{aligned} s_1 &= s_2 \\ \lambda_1 \cdot \mathbf{P_1} + (1 - \lambda_1) \cdot \mathbf{P_2} &= \lambda_2 \cdot \mathbf{P_3} + (1 - \lambda_2) \cdot \mathbf{P_4}. \end{aligned} \tag{9}$$

Equation 9 can be solved using the Mathematica code presented in Listing 4. This results in an expression for λ_1 (Equation 10) and one for λ_2 (Equation 11).

$$\lambda_1 = -\frac{-\mathbf{P_{2,x}}\mathbf{P_{3,y}} + \mathbf{P_{2,x}}\mathbf{P_{4,y}} + \mathbf{P_{2,y}}\mathbf{P_{3,x}} - \mathbf{P_{2,y}}\mathbf{P_{4,x}} - \mathbf{P_{3,x}}\mathbf{P_{4,y}} + \mathbf{P_{3,y}}\mathbf{P_{4,x}}}{q} \tag{10}$$

$$\lambda_2 = -\frac{-\mathbf{P_{1,x}}\mathbf{P_{2,y}} + \mathbf{P_{1,x}}\mathbf{P_{4,y}} + \mathbf{P_{1,y}}\mathbf{P_{2,x}} - \mathbf{P_{1,y}}\mathbf{P_{4,x}} - \mathbf{P_{2,x}}\mathbf{P_{4,y}} + \mathbf{P_{2,y}}\mathbf{P_{4,x}}}{q} \tag{11}$$

$$\begin{aligned} q = &-\mathbf{P_{1,x}}\mathbf{P_{3,y}} + \mathbf{P_{1,x}}\mathbf{P_{4,y}} + \mathbf{P_{1,y}}\mathbf{P_{3,x}} - \mathbf{P_{1,y}}\mathbf{P_{4,x}} + \mathbf{P_{2,x}}\mathbf{P_{3,y}} - \mathbf{P_{2,x}}\mathbf{P_{4,y}} \\ &- \mathbf{P_{2,y}}\mathbf{P_{3,x}} + \mathbf{P_{2,y}}\mathbf{P_{4,x}} \end{aligned} \tag{12}$$

q is the magnitude of the cross product of the vectors $\mathbf{v_1} = \mathbf{P_2} - \mathbf{P_1}$ and $\mathbf{v_2} = \mathbf{P_4} - \mathbf{P_3}$ when the vectors $\mathbf{P_1}$ through $\mathbf{P_4}$ are extended to three-dimensional space. If q is zero the vectors $\mathbf{v_1}$ and $\mathbf{v_2}$ are parallel and the two line segments will thus never intersect. If they are not parallel the two line segments only intersect when $\lambda_1, \lambda_2 \in [0, 1]$.

The Implementation

Based on the presented equations we have defined the method `line_segments_intersect` that takes two line segements defined by their endpoints and return `False` if they do not intersect and the intersection point if they do intersect. The code of that method is presented in Listing 5.

Projection of a Point on a Plane

To find the projection of a point $\mathbf{P_0}$ on a plane A defined by $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_3}$ we need to define the projection of the point and the plane in such a way that we can find an intersection.

The Plane

We can define the plane defined by the points $\mathbf{P_1}$, $\mathbf{P_2}$ and $\mathbf{P_3}$ by using the fact that the dot product of two vectors is zero if they are perpendicular. Thus a vector $\mathbf{q}$ is in the plane iff:

$$\mathbf{q} \cdot \mathbf{n} = 0. \tag{13}$$

Listing 5: The method `line_segments_intersect()`.

```
"""Module with methods that handle things related to line segments."""
from __future__ import division


def line_segments_intersect(segment_1, segment_2):
    """
    Return the point of intersection of segment one and two or none.

    Input:
        segment: List of two points, where each point is a list
            with the x and y coordinate of an endpoint of the line segment.
    """
    [p1, p2] = segment_1
    [p3, p4] = segment_2
    q = (
        -(p1[1]*p3[0]) + p2[1]*p3[0] + p1[0]*p3[1] - p2[0]*p3[1]
        + p1[1]*p4[0] - p2[1]*p4[0] - p1[0]*p4[1] + p2[0]*p4[1]
    )
    if(q):
        lambda_1 = (
            (p2[1] * p3[0] - p2[0] * p3[1] - p2[1] * p4[0] + p3[1] * p4[0]
                + p2[0] * p4[1] - p3[0] * p4[1]) / q
        )
        if(lambda_1 >= 0 and lambda_1 <= 1):
            lambda_2 = (
                - (p1[1] * p2[0] - p1[0] * p2[1] - p1[1] * p4[0] + p2[1] * p4[0]
                    + p1[0] * p4[1] - p2[0] * p4[1]) / q
            )
            if(lambda_2 >= 0 and lambda_2 <= 1):
                x = lambda_1 * p1[0] + (1 - lambda_1) * p2[0]
                y = lambda_1 * p1[1] + (1 - lambda_1) * p2[1]
                return [
                    lambda_1 * p1[0] + (1 - lambda_1) * p2[0],
                    lambda_1 * p1[1] + (1 - lambda_1) * p2[1]
                ]
    return None
```

Where $\mathbf{n}$ is the normal of the plane, which is defined as:

$$\mathbf{n} = (\mathbf{P_1} - \mathbf{P_2}) \times (\mathbf{P_3} - \mathbf{P_1}). \tag{14}$$

To determine if a point $\mathbf{P}$ lies in the plane we define a vector through that point that lies in the plane, thus we find that all points $\mathbf{P}$ lie in the plane iff:

$$(\mathbf{P} - \mathbf{P_1}) \cdot \mathbf{n} = 0 \tag{15}$$

The Projection Vector

Since the projection of the point $\mathbf{P_0}$, $\mathbf{P_0}'$ only has a different z-coordinate, we know that we can reach the projection point by starting at $\mathbf{P_0}$ and moving along the vector $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$. The line through $\mathbf{P_0}$ and the projection $\mathbf{P_0}'$ of that point is thus defined as:

$$\mathbf{P_0} + \lambda \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{16}$$

The Projection

Based on the definition of the plane (15) and the projection vector (16) we need to solve the following equation to find the projection:

$$\left(\left(\mathbf{P_0} + \lambda \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) - \mathbf{P_1} \right) \cdot \mathbf{n} = 0 \tag{17}$$

Listing 6: Mathematica code used to solve Equation 17.

```
p0 = {p0x, p0y, 0};
p1 = {p1x, p1y, p1z};
p2 = {p2x, p2y, p2z};
p3 = {p3x, p3y, p3z};
zvec = {0, 0, 1};

n = Cross[(p2 - p1), (p3 - p1)];
Solve[((p0 + lambda * zvec) - p1) . n == {0, 0, 0}, lambda]
```

The point $\mathbf{P_0}'$ is then found by filling the computed λ in into Equation 16. Since the z-coordinate of $\mathbf{P_0}$ is zero and the third element of the vector $\begin{bmatrix}0 & 0 & 1\end{bmatrix}$ is one, the z-coordinate of the projection point is λ.

Equation 17 can be solved in Mathematica using the code presented in Listing 6. The formula for λ is then:

$$\lambda = \frac{\begin{array}{l}\mathbf{P_{0,x}P_{1,y}P_{2,z}} - \mathbf{P_{0,x}P_{1,y}P_{3,z}} - \mathbf{P_{0,x}P_{1,z}P_{2,y}} + \mathbf{P_{0,x}P_{1,z}P_{3,y}} + \mathbf{P_{0,x}P_{2,y}P_{3,z}} - \mathbf{P_{0,x}P_{2,z}P_{3,y}} - \\ \mathbf{P_{0,y}P_{1,x}P_{2,z}} + \mathbf{P_{0,y}P_{1,x}P_{3,z}} + \mathbf{P_{0,y}P_{1,z}P_{2,x}} - \mathbf{P_{0,y}P_{1,z}P_{3,x}} - \mathbf{P_{0,y}P_{2,x}P_{3,z}} + \mathbf{P_{0,y}P_{2,z}P_{3,x}} - \\ \mathbf{P_{1,x}P_{2,y}P_{3,z}} + \mathbf{P_{1,x}P_{2,z}P_{3,y}} + \mathbf{P_{1,y}P_{2,x}P_{3,z}} - \mathbf{P_{1,y}P_{2,z}P_{3,x}} - \mathbf{P_{1,z}P_{2,x}P_{3,y}} + \mathbf{P_{1,z}P_{2,y}P_{3,x}}\end{array}}{-\mathbf{P_{1,x}P_{2,y}} + \mathbf{P_{1,x}P_{3,y}} + \mathbf{P_{1,y}P_{2,x}} - \mathbf{P_{1,y}P_{3,x}} - \mathbf{P_{2,x}P_{3,y}} + \mathbf{P_{2,y}P_{3,x}}} \tag{18}$$

If the numerator is zero the plane A is the x, y-plane. If the denominator is zero, the plane A is perpendicular to the project vector, and thus there is no projection point or there are infinitely many projection points.

The Implementation

Using the formula presented in Equation 18 we can define the function `project_point_on_plane ([p1, p2, p3], p0)` that computes the projection of the point `p0` on the plane defined by `p1, p2, p3`, see Listing 7.

C

To find the intersected edges and compute the intersection points we have used the method `find_intersected_edges`, see Listing 8. This method calls the method `line_segments_intersect` (Listing 5) on the edge defined by the points `p0` and `p1` and the edges of the triangulation. If an intersection is found it adds the indices of the intersected edge in the global list `intersected_line_segments` and appends the intersection point to `intersection_points`. The intersected edges of the triangulation and the intersection points are presented in Table 1.

To visualize the results we have added the code in Listing 9, the resulting visualization is shown in Figure 3.

Find and print the consecutive intersection points going from p0 to p1 without using the actual coordinates of the intersection points.

Listing 7: The method `project_point_on_plane()`.

```
"""Module with methods that handle things related to planes."""
from __future__ import division


def project_point_on_plane(plane, point):
    """
    Return False or the projection of point on plane.

    Input:
        plane: List of three points, where each point is a list
            with the x, y and z coordinate of a point that defines the plane.
        point: List with the x and y coordinate of the point.
    """
    [p1, p2, p3] = plane
    denominator = (
        p1[1] * p2[0] - p1[0] * p2[1] - p1[1] * p3[0] +
        p2[1] * p3[0] + p1[0] * p3[1] - p2[0] * p3[1]
    )
    if(denominator):
        numerator = (
            point[1] * p1[2] * p2[0] - point[0] * p1[2] * p2[1] -
            point[1] * p1[0] * p2[2] + point[0] * p1[1] * p2[2] -
            point[1] * p1[2] * p3[0] + p1[2] * p2[1] * p3[0] +
            point[1] * p2[2] * p3[0] - p1[1] * p2[2] * p3[0] +
            point[0] * p1[2] * p3[1] - p1[2] * p2[0] * p3[1] -
            point[0] * p2[2] * p3[1] + p1[0] * p2[2] * p3[1] +
            point[1] * p1[0] * p3[2] - point[0] * p1[1] * p3[2] -
            point[1] * p2[0] * p3[2] + p1[1] * p2[0] * p3[2] +
            point[0] * p2[1] * p3[2] - p1[0] * p2[1] * p3[2]
        )
        return [point[0], point[1], numerator/denominator]
    return False
```

Listing 8: The method `find_intersected_edges()`.

```
def find_intersected_edges():
    """
    Find the edges of the triangulation that are intersected by the linesegment p0-p1.

    The intersected edges are stored as an index in the lists xl and yl
    in the global variable intersected_line_segments.
    The intersection points are stored as list of length two in the
    global parameter intersection_points.
    """
    global intersected_line_segments, intersection_points, edgs, xl, yl, po, p1
    segment_1 = [p0, p1]
    for edge in edgs:
        segment_2 = [
            [xl[edge[0]], yl[edge[0]]],
            [xl[edge[1]], yl[edge[1]]]
        ]
        intersection = line_segments_intersect(segment_1, segment_2)
        if(intersection):
            intersected_line_segments.append(edge)
            intersection_points.append(intersection)
```

Listing 9: Part of the method `display()` that visualizes the intersected edges and the intersections.

```
    # draw intersected segments
    glColor3f(0.0, 0.0, 1)
    glBegin(GL_LINES)
    for edge in intersected_line_segments:
        glVertex2f(xl[edge[0]],  yl[edge[0]])
        glVertex2f(xl[edge[1]],  yl[edge[1]])
    glEnd()

    # draw intersection points on walk line
    glColor3f(0.0, 1.0, 0.0)
    glPointSize(4)
    glBegin(GL_POINTS)
    for point in intersection_points:
        glVertex2f(point[0], point[1])
    glEnd()
```

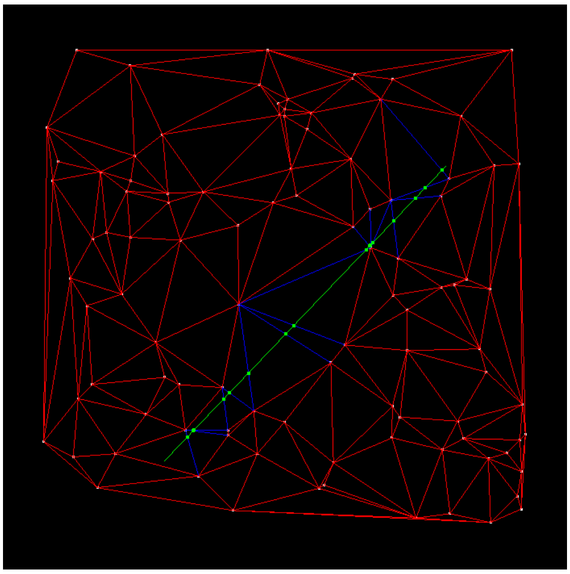

Figure 3: The visualization of the intersection of the line segment s with the edges of the triangulation dt. Each intersected edges is shown in blue, the green dots represent the intersections.

edge	intersection	edge	intersection
(2.26e2, 5.27e2) - (2.43e2, 5.84e2)	(2.29e2, 5.36e2)	(4.57e2, 3.01e2) - (2.93e2, 3.71e2)	(4.51e2, 3.04e2)
(2.26e2, 5.27e2) - (2.80e2, 5.34e2)	(2.36e2, 5.28e2)	(4.35e2, 2.75e2) - (4.57e2, 3.01e2)	(4.55e2, 2.99e2)
(2.26e2, 5.27e2) - (2.79e2, 5.28e2)	(2.37e2, 5.27e2)	(5.43e2, 2.37e2) - (4.81e2, 2.43e2)	(5.12e2, 2.40e2)
(2.72e2, 4.75e2) - (2.79e2, 5.28e2)	(2.74e2, 4.89e2)	(4.81e2, 2.43e2) - (4.91e2, 3.14e2)	(4.85e2, 2.68e2)
(2.72e2, 4.75e2) - (3.11e2, 5.03e2)	(2.81e2, 4.81e2)	(4.81e2, 2.43e2) - (4.57e2, 3.01e2)	(4.59e2, 2.95e2)
(2.93e2, 3.71e2) - (4.07e2, 4.44e2)	(3.51e2, 4.08e2)	(4.56e2, 2.53e2) - (4.57e2, 3.01e2)	(4.56e2, 2.98e2)
(2.93e2, 3.71e2) - (3.11e2, 5.03e2)	(3.05e2, 4.57e2)	(5.54e2, 2.16e2) - (4.81e2, 2.43e2)	(5.24e2, 2.27e2)
(2.93e2, 3.71e2) - (4.25e2, 4.22e2)	(3.61e2, 3.98e2)	(4.69e2, 1.17e2) - (5.54e2, 2.16e2)	(5.45e2, 2.05e2)

Table 1: The edges that were intersected by the line segment between `p0` and `p1` and the point where the line segment intersected the edge. A point $\mathbf{P}$ defined by its x and y coordinate is represented as (x, y). A linesegment between the points $\mathbf{P_1}$ and $\mathbf{P_2}$ is represented as $(\mathbf{P_1}, \mathbf{P_2})$.