Geometric Algorithms
Assignment 3

Laura Baakman (s1869140)

September 30, 2014

A

B

Line Segment Intersection

To find the intersection of the following two line segments

$$s_1 = \lambda_1 \cdot \mathbf{P_1} + (1 - \lambda_1) \cdot \mathbf{P_2} \qquad \text{for } 0 \leq \lambda_1 \leq 1 \tag{1}$$
$$s_2 = \lambda_2 \cdot \mathbf{P_3} + (1 - \lambda_2) \cdot \mathbf{P_4} \qquad \text{for } 0 \leq \lambda_2 \leq 1 \tag{2}$$

we need to solve the equation:

$$\begin{aligned} s_1 &= s_2 \\ \lambda_1 \cdot \mathbf{P_1} + (1 - \lambda_1) \cdot \mathbf{P_2} &= \lambda_2 \cdot \mathbf{P_3} + (1 - \lambda_2) \cdot \mathbf{P_4}. \end{aligned} \tag{3}$$

Equation 3 can be solved using the Mathematica code presented in Listing 1. This results in an expression for λ_1 (Equation 4) and one for λ_2 (Equation 5).

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

$$\lambda_2 = -\frac{-\mathbf{P_{1,x}P_{2,y}} + \mathbf{P_{1,x}P_{4,y}} + \mathbf{P_{1,y}P_{2,x}} - \mathbf{P_{1,y}P_{4,x}} - \mathbf{P_{2,x}P_{4,y}} + \mathbf{P_{2,y}P_{4,x}}}{q} \tag{5}$$

$$\begin{aligned} q = &-\mathbf{P_{1,x}P_{3,y}} + \mathbf{P_{1,x}P_{4,y}} + \mathbf{P_{1,y}P_{3,x}} - \mathbf{P_{1,y}P_{4,x}} + \mathbf{P_{2,x}P_{3,y}} - \mathbf{P_{2,x}P_{4,y}} \\ &- \mathbf{P_{2,y}P_{3,x}} + \mathbf{P_{2,y}P_{4,x}} \end{aligned} \tag{6}$$

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 $\lambda_1, \lambda_2 \in [0, 1]$.

Listing 1: Mathematica code used to solve Equation 3.

```
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}]
```

Listing 2: 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
```

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 2.

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:

$$\mathbf{q} \cdot \mathbf{n} = 0. \tag{7}$$

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

$$\mathbf{n} = (\mathbf{P_1} - \mathbf{P_2}) \times (\mathbf{P_3} - \mathbf{P_1}). \tag{8}$$

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:

$$(\mathbf{P} - \mathbf{P_1}) \cdot \mathbf{n} = 0 \tag{9}$$

Listing 3: Mathematica code used to solve Equation 11.

```
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 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:

$$\mathbf{P_0} + \lambda \cdot \begin{bmatrix}0\\0\\1\end{bmatrix} \tag{10}$$

The Projection

Based on the definition of the plane (9) and the projection vector (10) we need to solve the following equation to find the projection:

$$\left(\left(\mathbf{P_0} + \lambda \cdot \begin{bmatrix}0\\0\\1\end{bmatrix}\right) - \mathbf{P_1}\right) \cdot \mathbf{n} = 0 \tag{11}$$

The point $\mathbf{P_0}'$ is then found by filling the computed λ in into Equation 10. 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 11 can be solved in Mathematica using the code presented in Listing 3. The formula for λ is then:

$$\lambda = \frac{\begin{matrix}\mathbf{P_{0,x}P_{1,y}P_{2,z}} - \mathbf{P_{0,x}P_{1,y}P_{3,z}} - \mathbf{P_{0,x}P_{1,z}P_{2,y}} + \mathbf{P_{0,x}P_{1,z}P_{3,y}} + \mathbf{P_{0,x}P_{2,y}P_{3,z}} - \mathbf{P_{0,x}P_{2,z}P_{3,y}} - \\ \mathbf{P_{0,y}P_{1,x}P_{2,z}} + \mathbf{P_{0,y}P_{1,x}P_{3,z}} + \mathbf{P_{0,y}P_{1,z}P_{2,x}} - \mathbf{P_{0,y}P_{1,z}P_{3,x}} - \mathbf{P_{0,y}P_{2,x}P_{3,z}} + \mathbf{P_{0,y}P_{2,z}P_{3,x}} - \\ \mathbf{P_{1,x}P_{2,y}P_{3,z}} + \mathbf{P_{1,x}P_{2,z}P_{3,y}} + \mathbf{P_{1,y}P_{2,x}P_{3,z}} - \mathbf{P_{1,y}P_{2,z}P_{3,x}} - \mathbf{P_{1,z}P_{2,x}P_{3,y}} + \mathbf{P_{1,z}P_{2,y}P_{3,x}}\end{matrix}}{-\mathbf{P_{1,x}P_{2,y}} + \mathbf{P_{1,x}P_{3,y}} + \mathbf{P_{1,y}P_{2,x}} - \mathbf{P_{1,y}P_{3,x}} - \mathbf{P_{2,x}P_{3,y}} + \mathbf{P_{2,y}P_{3,x}}} \tag{12}$$

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 12 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 4.

Listing 4: 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
```