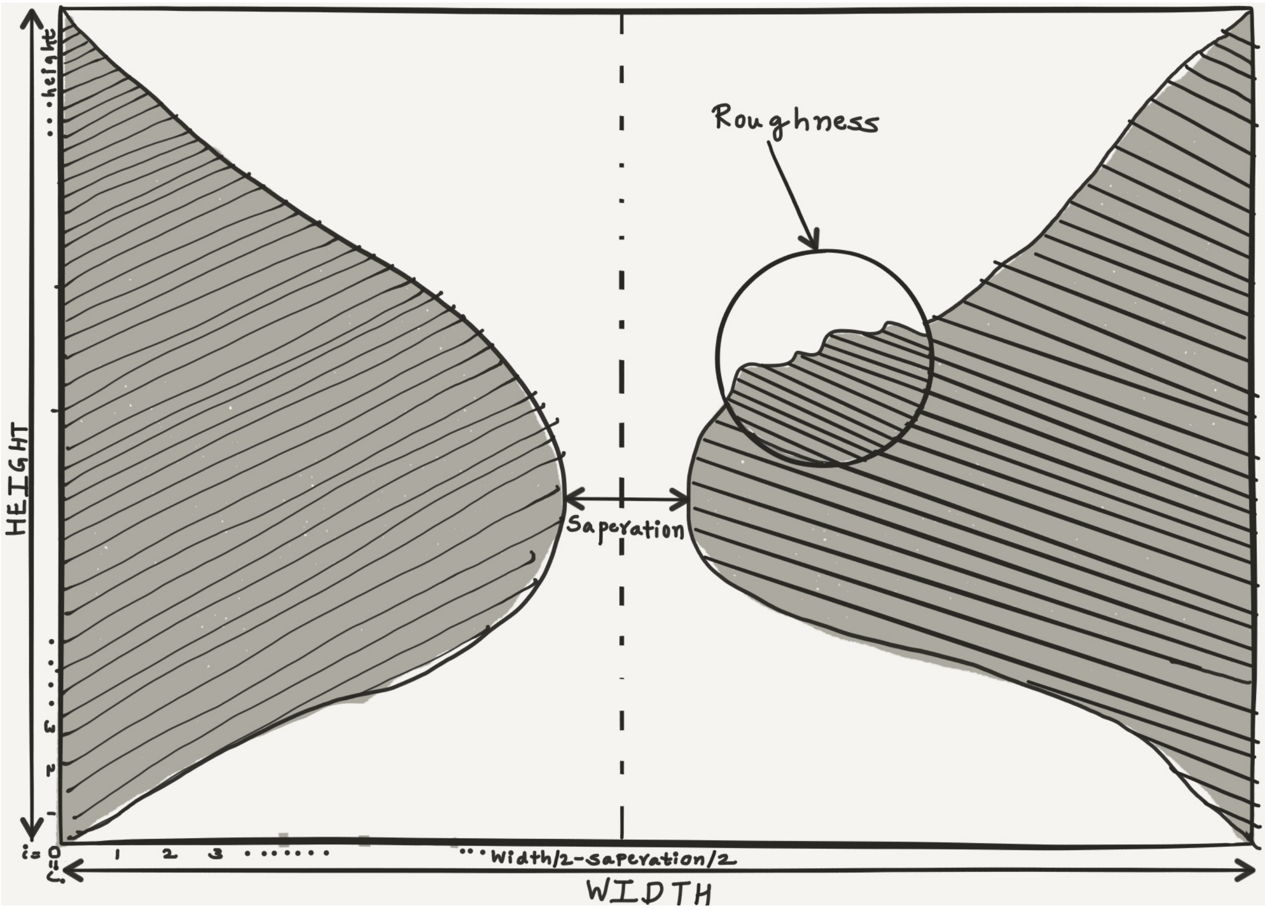
Terrain generation and roughness

The terrain consists of two surfaces (left and right) separated by some distance. This can be done using a simple sine wave. The surface also has some roughness, which is modulated by the sine function using Perlin noise.



Pseudocode

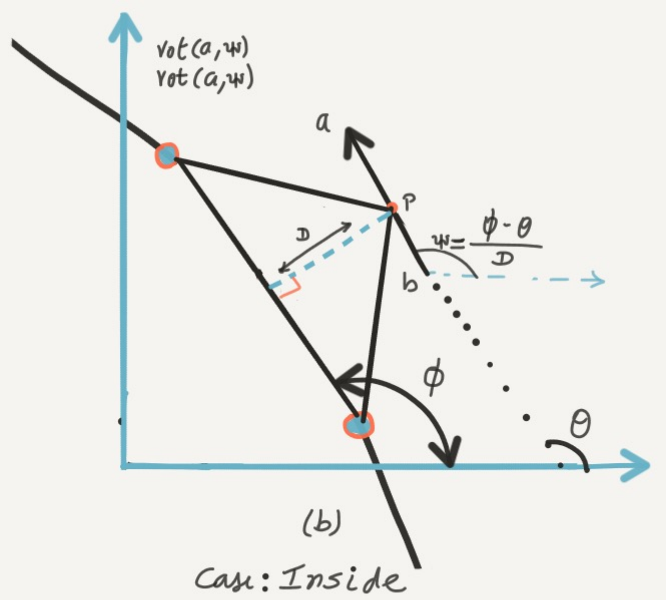
```
algorithm terrain generator is
    input: Height H of the window,
           Width W of the window,
           Saperation S between the surfaces,
           Roughness R of the surfaces,
           Resolution r for the number of points

    output: Vetices of right a and left b surfaces

 let y:= height
 let θ:=0
 let i:=0 be the iterator for vertices

While y is greater than zero do
  let xRight be the x coordinate of the right vertex
  let xLeft be the x coordinate of the left vertex
  xRight ← (map cos(θ) between S and W) + noise(R)
  xLeft ← (map cos(θ) between 0 and W-saperation) + noise(R)

  increment θ
  increment R

  a(i) ← [xRight y]
  b(i) ← [xLeft y]
  y ← y+r
}
```

Flow field

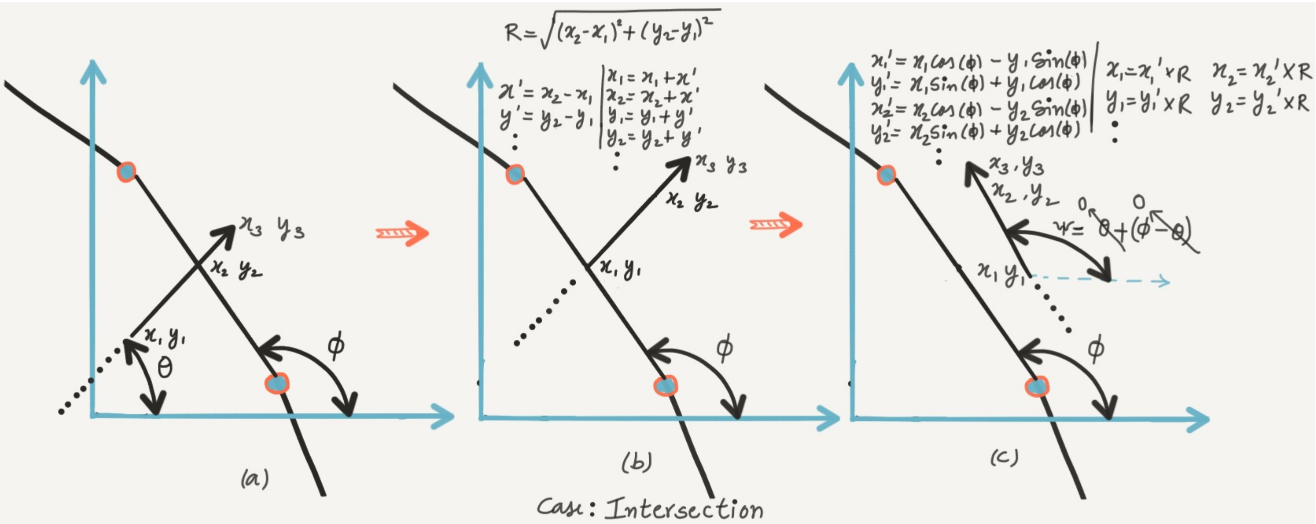
Perlin noise blah

Flow field around terrain based on terrain

When the flow field vector is inside the terrain following method is used



When the flow field vector is intersecting the terrain following is done.



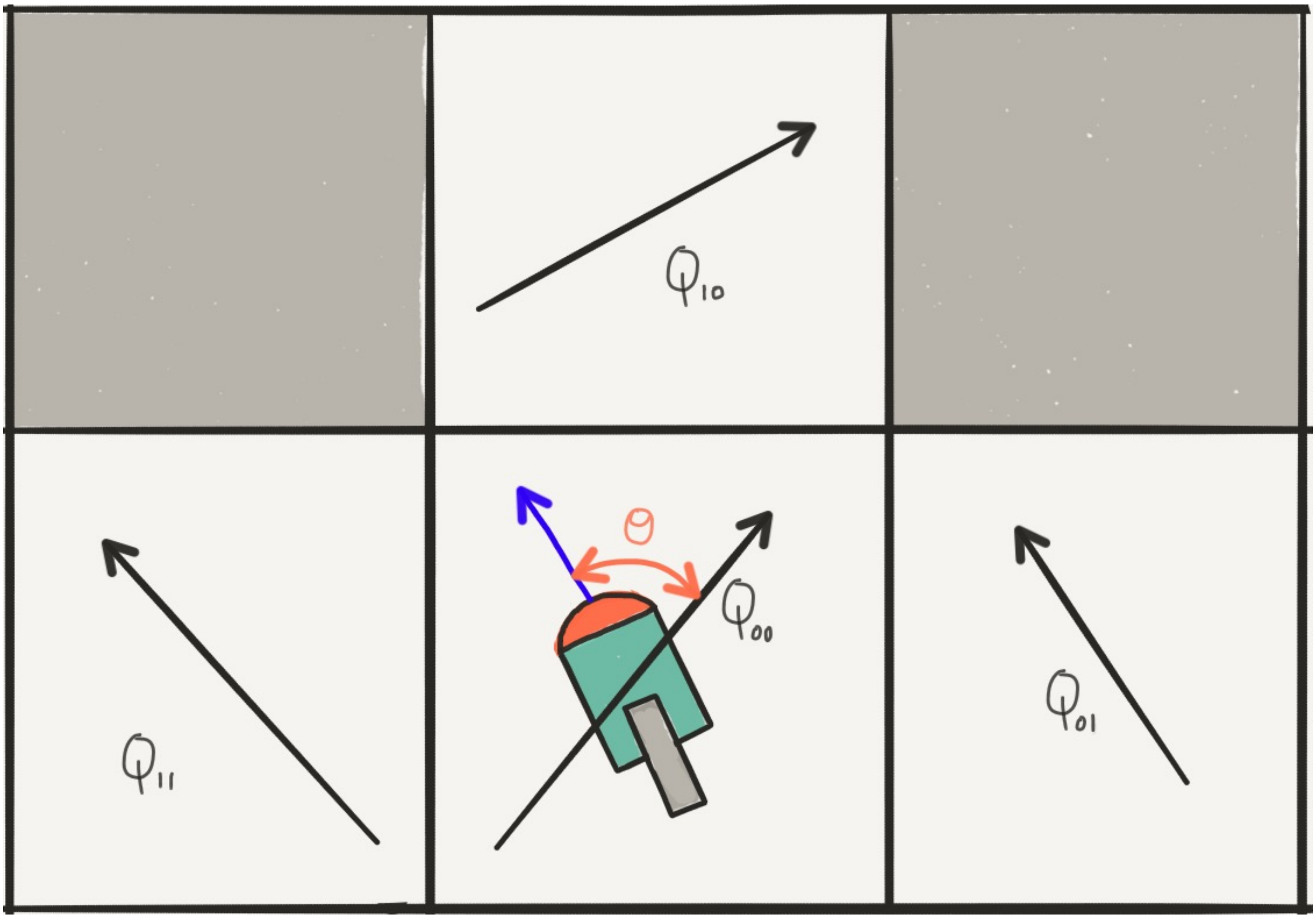
When the flow field vector is outside the terrain, its magnitude is set to zero.

Psuedocode

```
algorithm flow field generator is
    input: Noise xn along x axis ,
           Noise yn along y axis,
           Polygon P

    output: A two flowfield f

Let f be the flow field array,
Let xoff := 0 be the iterator for x axis,
Let c be the number of columns in f,
Let r be the number of rows in in r
Let θ := 0 be a theta value (for the angle of vector)
Let Φ := 0 be the inversion factor for θ which gets stronger as a vector get closer to a boundary

While i is less than number of columns
    Let yoff := 0 be the iterator for y axis
    while j is less than number of rows
        θ ← map noise(xoff,yoff) between 0 and -π;
        f(i,j) ← (cos(θ), sin(θ)) // Polar to cartesian coordinate transformation
        Let A, B be two nearest points in P to the above vector.
        Φ ← atan2((By - Ay),(Bx - By))

        if f(i,j) lies outside P
            f(i,j) ← (0,0)

        if f(i,j) lies inside P
            Let D be the perpendicular distance between the center of f(i,j) and
            the line segment AB

            Let Ψ ← (Φ - θ)/D
            f(i,j) ← Rotate f(i,j) by Ψ

        if f(i,j) intersects P
            Let D be the intersection point
            Let A and B the endpoints of the vector formed by f(i,j)
            Let d ← (Bx-Dx) (By-Dy) be the translation vector
            f(i,j) ← translate(d)
            Let Ψ ← Φ
            f(i,j)  ← rotate(Φ)
        yoff = yoff + yn
        iterate i
    xoff = xoff + xn
    iterate j

    return f
```

Flowfield following

Using the flow field, we need to apply forces on the robot. To follow the Flow Field we will implement a steering behavior that directs the agent in the direction of the grid square it is standing on. To smooth this vector we will use Bilinear Interpolation so we get influenced by the 4 grid squares we are nearest to, with the closest providing the most influence. This seems better than just averaging out the neighboring vectors.



Since the flow field array is absolute, we need to floor it before querying the flow field vector.

$f(x, y_1) = f(Q_{00})(1 - x + Q_{00}x) + f(Q_{10} * (x - Q_{00}))$

$f(x, y_2) = f(Q_{01})(1 - x + Q_{00}x) + f(Q_{11} * (x - Q_{00}))$

Secondly using y-axis interpolation the desired direction can be given as

$\vec{d} = \| f(x, y_1)(1 - y + Q_{00}y) + f(x, y_2)(y - Q_{00}y) \|$

Now we can calculate the force which needs to be applied to the robot. The desired velocity is simply

$\vec{v} = V \times \vec{d}$ **where** V **is a constant speed**

There for the change in velocity needed is

$\delta\vec{v} = \vec{v} - \vec{u}$ **where** $\vec{u}$ **is the current speed**

Finally, the force to be applied on the body is

$\vec{f} = \delta\vec{v}(\frac{F}{V})$ **where F is a constant force**

Although this force applied at the robot's center, will push the robot in the right direction, it will not keep the robot aligned in the right direction. A simple way to do this is to simply rotate the robot by applying a transform.

Notice that $\phi = \tan^{-1}(\frac{-\vec{d}_x}{\vec{d}_y})$ is the desired angle of rotation

However, this rotating the body like this is

1. Not exactly physics
2. Is jittery

Notice that the transformation applied earlier is equivalent to applying the exact torque to the body-center. Therefore, the jitter occurs because the exact torque causes the robot to move past ϕ . Instead the torque should decrease as $\theta \rightarrow \phi$ **where is the current angle** (the body reached the correct alignment). In order to do this, we need to calculate the angle in the future time steps (more time step we look ahead, better the approximation). Thus,

$\theta_{t+1} = \theta_t + \dot{\theta} \times t$ **where t is time ahead at which we are calculating the new theta**

So the small desired rotation we need to apply at time t is

$\delta\theta = \phi - \theta_{t+1}$

so the angular velocity then is

$\delta\dot{\theta} = \frac{\delta\theta}{t}$

finally the toque, $\tau = \frac{I \times \delta\dot{\theta}}{t}$ where I is the rotational inertia

---