# Creating a Simple Vector Calculus Framework to Verify Stokes' Curl Theorem

Mohamed Hussien El-Deeb (201900052)

May 12, 2024

## Contents

## 1 Introduction

Stokes' curl theorem, also known as the Kelvin-Stokes theorem after Lord Kelvin and George Stokes, is a theorem in vector calculus on $\mathbb{R}^3$. Given a vector field, the theorem relates the integral of the curl of the vector field over some surface, to the line integral of the vector field around the boundary of the surface. This theorem can be stated as: The line integral of a vector field over a loop is equal to the surface integral of its curl over the enclosed surface. This unlocks integration shortcuts that greatly simplifies other theorems and speeds up computations. This project is an attempt to build a simple vector calculus framework to verify and visualize Stokes' curl theorem using basic numerical methods and the web graphics library three.js.

# 2 Stokes' Curl Theorem

## 2.1 Definition

Stokes' theorem states that the line integral of a vector field $\mathbf{F}$ around a closed curve $C$ is equal to the surface integral of the curl of $\mathbf{F}$ over any surface $S$ bounded by $C$. Mathematically, the theorem can be stated as:

$$\oint_C \mathbf{F} \cdot d\mathbf{r} = \iint_S (\nabla \times \mathbf{F}) \cdot d\mathbf{S}$$

where:

- $\mathbf{F}$ is a differentiable vector field.
- $C$ is a piecewise-smooth, positively oriented simple closed curve in space.
- $S$ is a piecewise-smooth, positively oriented surface in space bounded by $C$.
- $d\mathbf{r}$ is the differential arc vector along $C$.
- $d\mathbf{S}$ is the differential area vector of $S$.

# 3 Integration over a Path

## 3.1 Definition

Since computers are discrete, we need to approximate the integral of a function over a path using numerical methods which means we will revert to using summation instead of integration while noting the values converge as the step size decreases.

Suppose we want to sum up the values of a function $f(x, y, z)$ along a path $C$ in space.

$$C = \{P(t) = \langle x(t), y(t), z(t)\rangle \mid 0 \leq t \leq 1\}$$

We can approximate the path by dividing it into many small line segments and approximate the function by a constant value over each segment. The sum of the values of the function over all segments is the integral of the function over the path. Numerically, the integral can be approximated by the following formula:

$$\sum_{i=0}^{n} f(L_i)$$

where:

- $f$ is the function to be integrated.
- $L_i$ is a small line segment on the path.
- $n$ is the integer number of line segments.

$L_i$ is defined as follows:

$$L_i = [C\left(i\Delta t\right), C\left((i+1)\Delta t\right)]$$

where:

- $C(t)$ is the parametric equation of the path.
- $\Delta t$ is the step size.

We can choose our parameter $t$ to be $0 \leq t \leq 1$ meaning we can substitute $\Delta t$ with $\frac{1}{n}$.

Note that it is possible to convert parameter limits from $[a, b]$ to $[0, 1]$ using the following formula as long as $a \neq b$:

$$P(t), \quad a \leq t \leq b \quad \rightarrow \quad P\left(\frac{t-a}{b-a}\right), \quad 0 \leq t \leq 1$$

Now we arrive at the following formula:

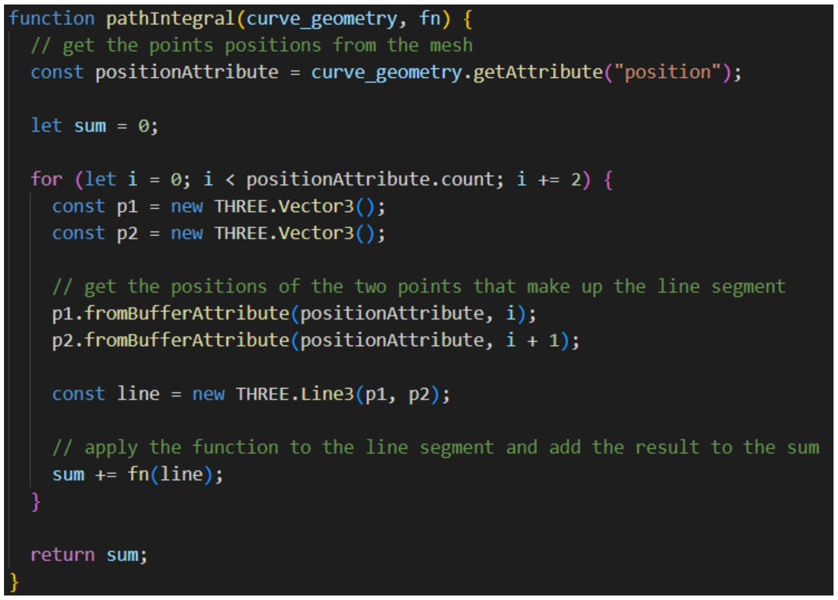
$$\sum_{i=0}^{n} f\left(L_i\right)$$

$$L_i = \left[C\left(\frac{i}{n}\right), C\left(\frac{i+1}{n}\right)\right]$$

The higher the value of $n$, the more accurate the approximation.

## 3.2 Implementation

As a proof of concept, we can implement the path integral of a scalar field in three.js[2]. All example code is part of the implementation software and can be found in the repository[1]. Note that knowledge of three.js and computer graphics is required to understand the code.


```
function pathIntegral(curve_geometry, fn) {
  // get the points positions from the mesh
  const positionAttribute = curve_geometry.getAttribute("position");

  let sum = 0;

  for (let i = 0; i < positionAttribute.count; i += 2) {
    const p1 = new THREE.Vector3();
    const p2 = new THREE.Vector3();

    // get the positions of the two points that make up the line segment
    p1.fromBufferAttribute(positionAttribute, i);
    p2.fromBufferAttribute(positionAttribute, i + 1);

    const line = new THREE.Line3(p1, p2);

    // apply the function to the line segment and add the result to the sum
    sum += fn(line);
  }

  return sum;
}
```

Figure 1: Path Integral of a Scalar Field Code[2]

As it can be seen, the path integral of a scalar field is approximated by summing up the values of the scalar field over many small line segments on the path. The boundary path itself is constructed using the edge geometry in three.js.

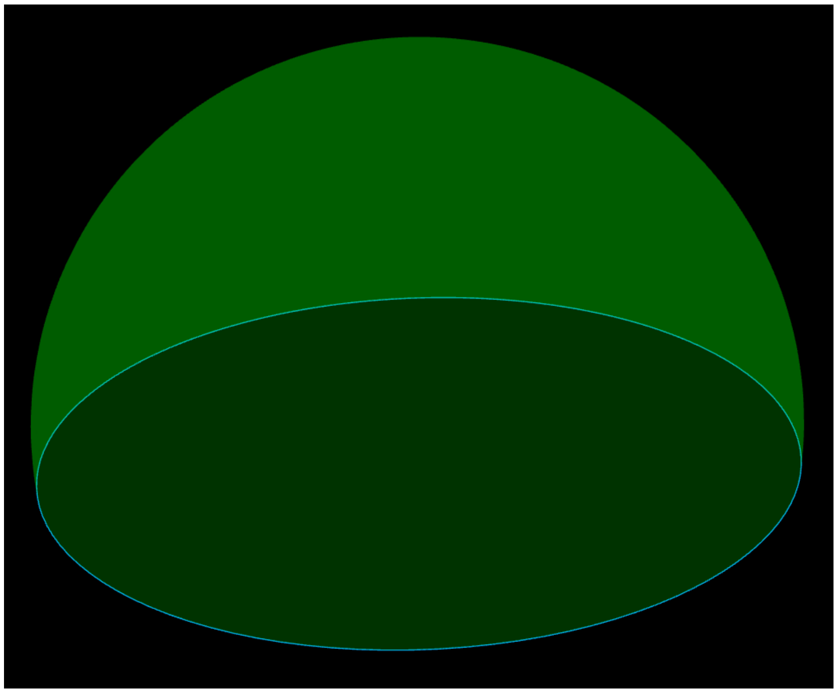


Figure 2: boundary path of the surface of half a sphere[2]

This integration method is used to compute the path integral side of Stokes' curl theorem.

# 4 Integration over a Surface

## 4.1 Definition

Similarly, we can approximate the integral of a function $f(x, y, z)$ over a surface $S$ using numerical methods. We can divide the surface into small triangles and approximate the function by a constant value over each triangle. The sum of the values of the function over all triangles converges to the integral of the function over the surface as the limit of the step size reach 0.

To construct our triangle mesh, we can use the parametric equation of the surface $S(u, v)$ where $0 \leq u \leq 1$ and $0 \leq v \leq 1$. and sample the surface at $n \times m$ points.

$$S(u, v) = [x(u, v), y(u, v), z(u, v)]$$

$$T_{ij} = [S\left(i\Delta u, j\Delta v\right), S\left((i+1)\Delta u, j\Delta v\right), S\left(i\Delta u, (j+1)\Delta v\right)]$$

For the sake of simplicity, we can assume $\Delta u = \Delta v = \frac{1}{n}$.

$$T_{ij} = \left[S\left(\frac{i}{n}, \frac{j}{n}\right), S\left(\frac{i+1}{n}, \frac{j}{n}\right), S\left(\frac{i}{n}, \frac{j+1}{n}\right)\right]$$

Note that this only captures half of the surface, to capture the other half we can use the following formula:

$$T_{ij} = \left[S\left(\frac{i+1}{n}, \frac{j}{n}\right), S\left(\frac{i}{n}, \frac{j+1}{n}\right), S\left(\frac{i+1}{n}, \frac{j+1}{n}\right)\right]$$

Now we can approximate the integral of the function over the surface using the following formula similar to the path integral formula:

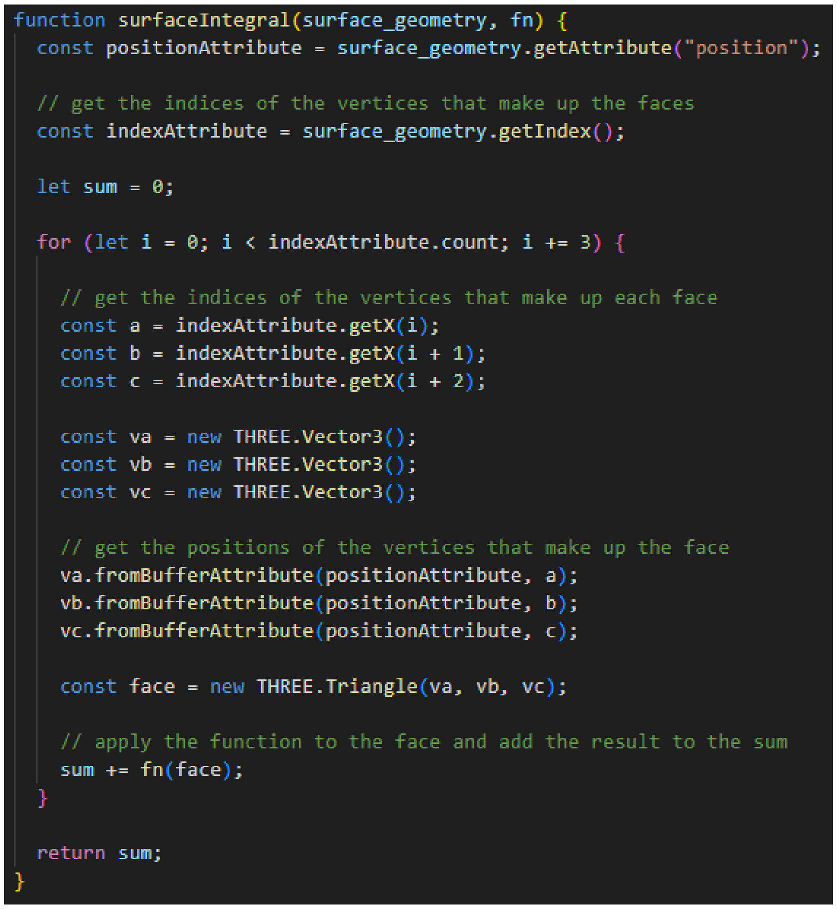
$$\sum_{i,j=0}^{n} f(T_{ij})$$

where:

- $f$ is the function to be integrated.
- $T_{ij}$ is a small triangle on the surface.
- $n$ is the integer number of steps per dimension.

## 4.2 Implementation

Surface integration follows the same spirit of path integration, we can implement the surface integral of a scalar field in three.js[2]. Instead of constructing a boundary path, we construct a surface mesh using parametric geometry in three.js and loop over the triangles within.



```
function surfaceIntegral(surface_geometry, fn) {
  const positionAttribute = surface_geometry.getAttribute("position");

  // get the indices of the vertices that make up the faces
  const indexAttribute = surface_geometry.getIndex();

  let sum = 0;

  for (let i = 0; i < indexAttribute.count; i += 3) {

    // get the indices of the vertices that make up each face
    const a = indexAttribute.getX(i);
    const b = indexAttribute.getX(i + 1);
    const c = indexAttribute.getX(i + 2);

    const va = new THREE.Vector3();
    const vb = new THREE.Vector3();
    const vc = new THREE.Vector3();

    // get the positions of the vertices that make up the face
    va.fromBufferAttribute(positionAttribute, a);
    vb.fromBufferAttribute(positionAttribute, b);
    vc.fromBufferAttribute(positionAttribute, c);

    const face = new THREE.Triangle(va, vb, vc);

    // apply the function to the face and add the result to the sum
    sum += fn(face);
  }

  return sum;
}
```

Figure 3: Surface Integral of a Scalar Field Code[2]

As it can be seen, the surface integral of a scalar field is approximated by summing up the values of the scalar field over many small triangles on the surface.

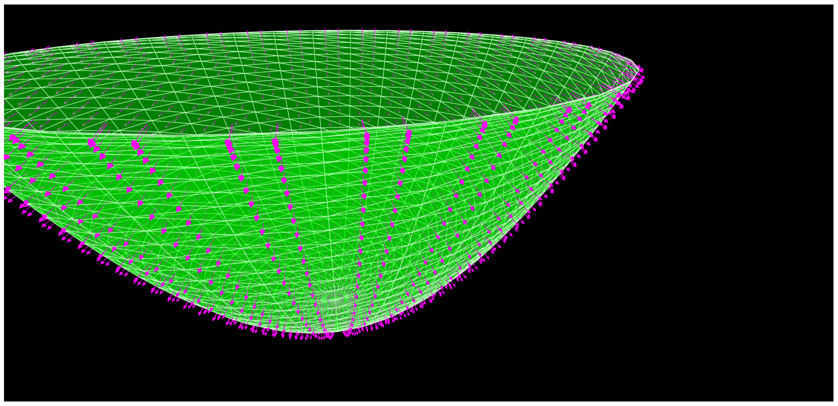


Figure 4: Triangle mesh and surface normals of the surface of a paraboloid[2]

This integration method is used to compute the surface integral side of Stokes' curl theorem.

# 5 Scaler Field

## 5.1 Definition

A scalar field is a function that assigns a scalar value to every point in space. The value may be constant or vary over time and space. Mathematically, a scalar field can be represented as:

$$f : \mathbb{R}^3 \to \mathbb{R}$$

This is trivial to implement in code as it is just a function that takes a 3D point as input and returns a scalar value. the only issue is to be able to handle singular points where the function is not defined.

We may want to visualize the scalar field and numerous ways to do so, one way is to use a color map to represent the scalar value at each point. We can use the heat map color map where the color represents the value of the scalar field at each point. In that case we would need to pre-compute the scalar field to find the minimum and maximum values to normalize the color map. if the field is 2D we plot it on a 3D surface where the height of the surface represents the value of the scalar field at each point.

## 5.2 Implementation

The implementation of a scalar field is straightforward, we can implement the scalar field as a class that contains an arbitrary function and a method that takes a 3D point as input and returns a scalar value. And class methods to represent other operations on the scalar field such as the partial derivatives and the gradient.

## 5.3 Partial Derivatives of a Scalar Field

### 5.3.1 Definition

The partial derivatives of a scalar field are the derivatives of the scalar field with respect to each of the independent variables. The partial derivatives are defined as:

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}$$

We could implement the partial derivatives as functions that take a 3D point as input and return a scalar value. Where the partial derivatives can be approximated using finite differences.

$$\frac{\partial f}{\partial x} \approx \frac{f(x+h,y,z) - f(x-h,y,z)}{2h}$$

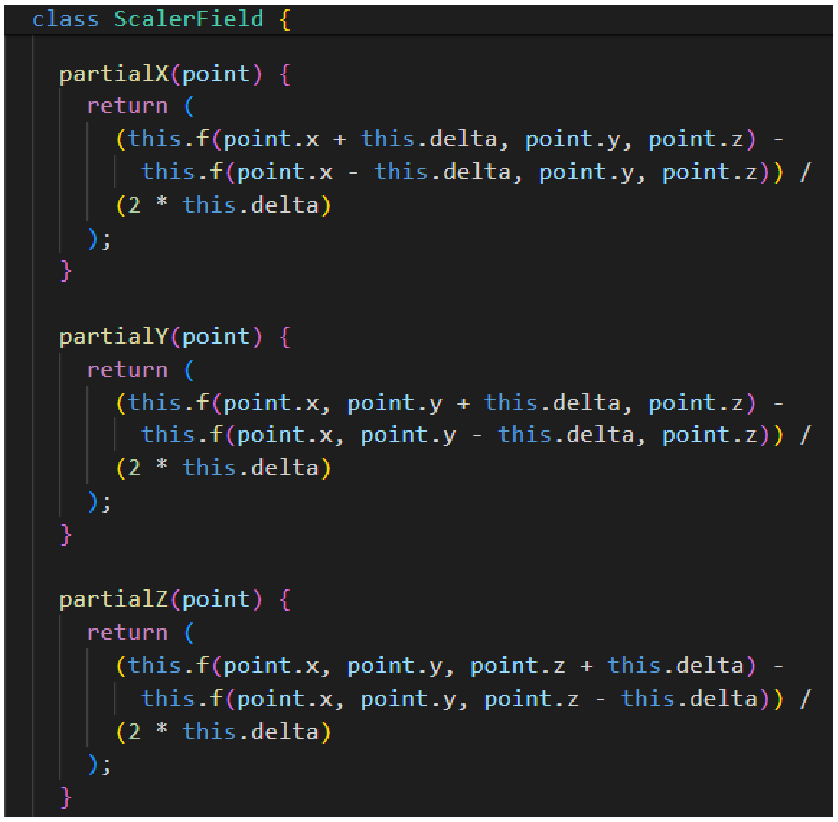
$$\frac{\partial f}{\partial y} \approx \frac{f(x,y+h,z) - f(x,y-h,z)}{2h}$$

$$\frac{\partial f}{\partial z} \approx \frac{f(x,y,z+h) - f(x,y,z-h)}{2h}$$

where $h$ is the step size. We need to be careful with the step size as it affects the accuracy of the approximation. We want to choose a small step size to get a more accurate approximation but not too small to avoid numerical errors.

### 5.3.2 Implementation

In this program, we can implement the partial derivatives using the finite difference method.



```
class ScalerField {

  partialX(point) {
    return (
      (this.f(point.x + this.delta, point.y, point.z) -
        this.f(point.x - this.delta, point.y, point.z)) /
      (2 * this.delta)
    );
  }

  partialY(point) {
    return (
      (this.f(point.x, point.y + this.delta, point.z) -
        this.f(point.x, point.y - this.delta, point.z)) /
      (2 * this.delta)
    );
  }

  partialZ(point) {
    return (
      (this.f(point.x, point.y, point.z + this.delta) -
        this.f(point.x, point.y, point.z - this.delta)) /
      (2 * this.delta)
    );
  }
```

Figure 5: Partial Derivatives of a Scalar Field Code[2]

Note: We evaluate the partial derivatives at the center of the cells to get a more accurate approximation.

## 5.4 Gradient of a Scalar Field

### 5.4.1 Definition

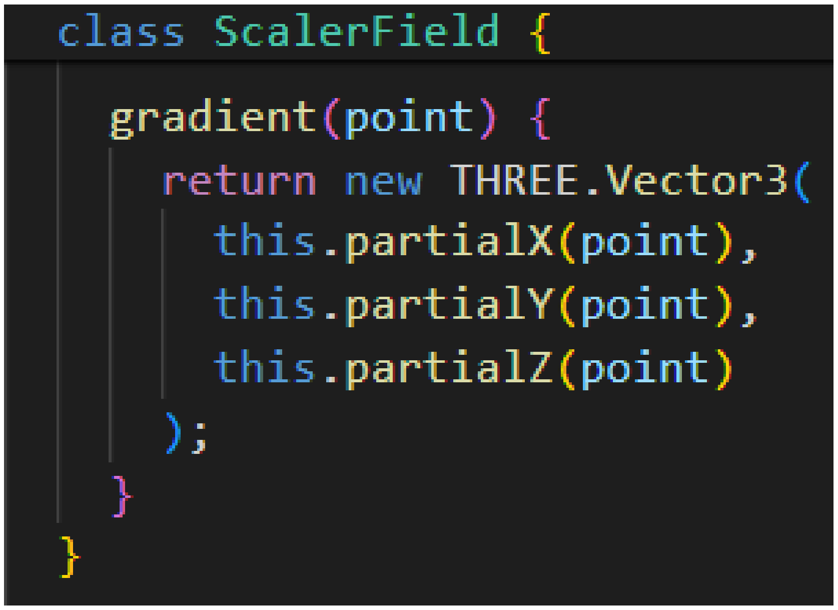
The gradient of a scalar field is a vector field that points in the direction of the greatest rate of increase of the scalar field at each point. The gradient is defined as:

$$\nabla f = \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right\rangle$$

We could implement the gradient as a function that takes a 3D point as input and returns a 3D vector. Where the partial derivatives can be approximated using finite differences.

### 5.4.2 Implementation

The gradient of a scalar field is the an implementation of the partial derivatives of the scalar field.



```
class ScalerField {
  gradient(point) {
    return new THREE.Vector3(
      this.partialX(point),
      this.partialY(point),
      this.partialZ(point)
    );
  }
}
```

Figure 6: Gradient of a Scalar Field Code[2]

# 6 Vector Field

## 6.1 Definition

A vector field is a function that assigns a vector to every point in space. The vector may be constant or vary over time and space. Mathematically, a vector field can be represented as:

$$\mathbf{F} : \mathbb{R}^3 \to \mathbb{R}^3$$

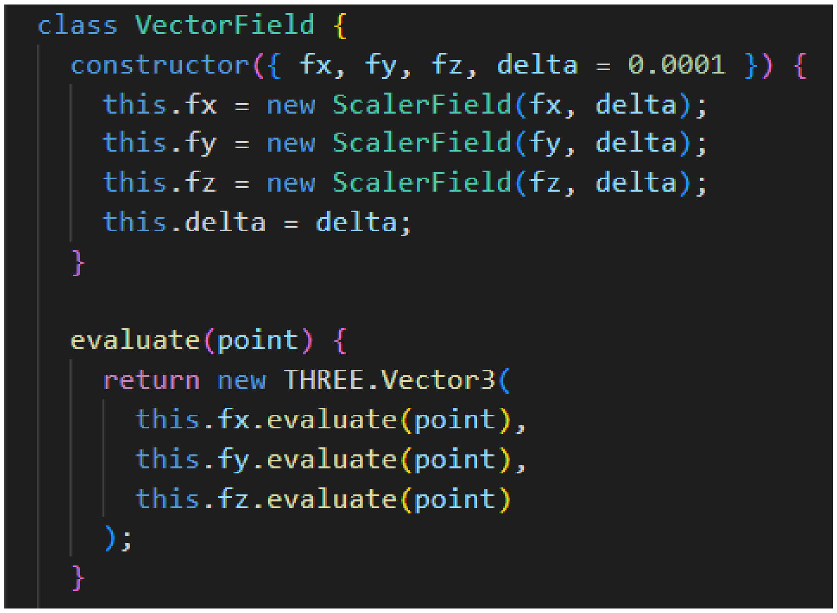
This is also trivial to implement in code as it is just a function that takes a 3D point as input and returns a 3D vector.

Note that vector fields can for all intents and purposes be treated as n scalar fields where n is the dimension of the vector field. The will come in when we discuss the divergence and curl of a vector field.

To visualize a vector field, we can use arrows to represent the vectors at each point. The length of the arrow represents the magnitude of the vector and the direction of the arrow represents the direction of the vector. We can also use color to represent the magnitude of the vector and in that case we could normalize the arrows to have the same length. There are two common way to visualize a vector field, the first is to plot the vectors starting from the sampled points and the second is to plot the vectors at the center of the cells. The difference only amounts to a shift in the position of the vectors by half its length parallel to its direction.

## 6.2 Implementation

The implementation of a vector field is similar to the implementation of a scalar field, we can implement the vector field as a class that contains 3 arbitrary functions and have class methods to represent other operations on the vector field such as the divergence and the curl.



```
class VectorField {
  constructor({ fx, fy, fz, delta = 0.0001 }) {
    this.fx = new ScalerField(fx, delta);
    this.fy = new ScalerField(fy, delta);
    this.fz = new ScalerField(fz, delta);
    this.delta = delta;
  }

  evaluate(point) {
    return new THREE.Vector3(
      this.fx.evaluate(point),
      this.fy.evaluate(point),
      this.fz.evaluate(point)
    );
  }
```

Figure 7: Vector Field Code[2]

Note: The evaluate function just evaluates the vector field at a given point by calling the functions and return the result.

## 6.3 Divergence of a Vector Field

### 6.3.1 Definition

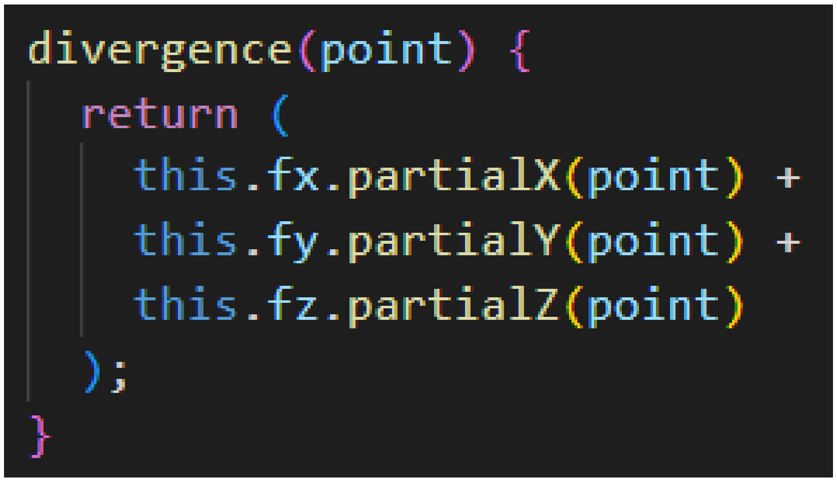
The divergence of a vector field is a scalar field that represents the rate at which the vector field is expanding at each point. The divergence is defined as:

$$\nabla \cdot \mathbf{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}$$

Since in divergence we are dealing with each component of the vector field separately, we could treat each component as a scalar field and we just compute the partial derivate of that component with respect to the corresponding axis. Then we sum up the partial derivatives of each component to get the divergence.

### 6.3.2 Implementation

The divergence of a vector field is the sum of the partial derivatives of each component of the vector field.



```
divergence(point) {
  return (
    this.fx.partialX(point) +
    this.fy.partialY(point) +
    this.fz.partialZ(point)
  );
}
```

Figure 8: Divergence of a Vector Field Code[2]

## 6.4 Curl of a Vector Field

### 6.4.1 Definition

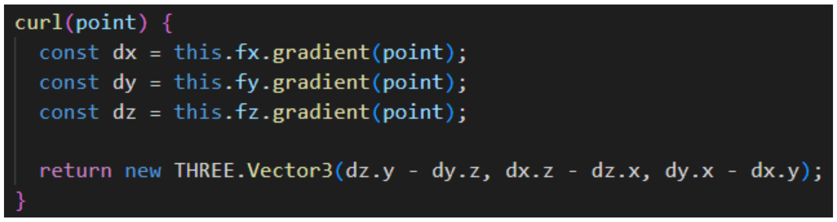
The curl of a vector field is a vector field that represents the rotation of the vector field at each point. The curl is defined as:

$$\nabla \times \mathbf{F} = \left\langle \frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z}, \frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x}, \frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y} \right\rangle$$

We can use the same approach as the divergence to compute the curl. We treat each component of the vector field as a scalar field and compute the partial derivatives of each component with respect to the specified axes. Then we arrange the partial derivatives in the correct order to get the curl.

### 6.4.2 Implementation

The curl of a vector field is the difference of the partial derivatives of each component of the vector field.



```
curl(point) {
  const dx = this.fx.gradient(point);
  const dy = this.fy.gradient(point);
  const dz = this.fz.gradient(point);

  return new THREE.Vector3(dz.y - dy.z, dx.z - dz.x, dy.x - dx.y);
}
```

Figure 9: Curl of a Vector Field Code[2]

# 7 Result

## 7.1 Verification of Stokes' Curl Theorem

There are many ways to verify the correctness of the implementation, one way is to compare the numerical results with the analytical solutions.

### 7.1.1 Example 1

A triangle with vertices $(0,0,3),(0,2,0)$ and $(4,0,0)$ in the $xy$-plane. The vector field is $\vec{F}=(3yx^2+z^3)\,\hat{\imath}+y^2\hat{\jmath}+4yx^2\hat{k}$. And the rotation is counter clockwise if you are above the triangle and looking towards the $xy$-plane.

The analytical solution is:

$$\iint_S \vec{\nabla}\times\overrightarrow{\boldsymbol{u}}\cdot\widehat{\boldsymbol{n}}dS=\oint_C \overrightarrow{\boldsymbol{u}}\cdot d\vec{r}$$

$$\vec{\nabla}\times\vec{F}=\begin{vmatrix}\hat{\imath} & \hat{\jmath} & \hat{k}\\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z}\\ 3yx^2+z^3 & y^2 & 4yx^2\end{vmatrix}=4x^2\hat{\imath}+\left(3z^2-8xy\right)\hat{\jmath}-3x^2\hat{k}$$

The normal of the triangle is same as the normal of the plane that passes through the three vertices of the triangle.

The equation of the plane is

$$\frac{x}{4}+\frac{y}{2}+\frac{z}{3}=1$$

The normal of the plane and the triangle is

$$\vec{n}=\left(\frac{1}{4},\frac{1}{2},\frac{1}{3}\right)$$

$$\hat{n}=\frac{\vec{n}}{|\vec{n}|}=\frac{\frac{1}{4}\hat{\imath}+\frac{1}{2}\hat{\jmath}+\frac{1}{3}\hat{k}}{\sqrt{\frac{1}{16}+\frac{1}{4}+\frac{1}{9}}}=\frac{3\hat{\imath}+6\hat{\jmath}+4\hat{k}}{\sqrt{61}}$$

$$\vec{\nabla}\times\vec{F}\cdot\hat{n}=\left(4x^2\hat{\imath}+\left(3z^2-8xy\right)\hat{\jmath}-3x^2\hat{k}\right)\cdot\frac{3\hat{\imath}+6\hat{\jmath}+4\hat{k}}{\sqrt{61}}$$

$$=\frac{12x^2+18z^2-48xy-12x^2}{\sqrt{61}}=\frac{18z^2-48xy}{\sqrt{61}}$$

$$\iint_S\vec{\nabla}\times\vec{F}\cdot\hat{n}dS=\frac{6}{\sqrt{61}}\iint_S\left(3z^2-8xy\right)dS$$

$$\iint_S f(x,y,z)dS=\iint_D f(x,y,g(x,y))|\vec{\nabla}\emptyset|dxdy$$

$$g(x,y)=3-\frac{3}{4}x-\frac{3}{2}y$$

$$\emptyset=z-3+\frac{3}{4}x+\frac{3}{2}y$$

$$\vec{\nabla}\emptyset=\left(\frac{3}{4},\frac{3}{2},1\right)$$

$$|\vec{\nabla}\emptyset|=\sqrt{\left(\frac{3}{4}\right)^2+\left(\frac{3}{2}\right)^2+1^2}=\sqrt{\frac{9}{16}+\frac{9}{4}+1}=\sqrt{\frac{9+36+16}{16}}$$

$$=\sqrt{\frac{61}{16}}=\frac{\sqrt{61}}{4}$$

$$\iint_S\vec{\nabla}\times\vec{F}\cdot\hat{n}dS=\frac{3}{2}\int_0^2\int_0^{4-2y}\left(\frac{27}{16}(x+2y-4)^2-8xy\right)dxdy$$

$$=\frac{3}{2}\int_0^2\left[\frac{27}{48}(x+2y-4)^3-4x^2y\right]_0^{4-2y}dy$$

$$=-\frac{3}{2}\int_0^2\left(16(y-2)^2y+\frac{27}{6}(y-2)^3\right)dy$$

$$= -\frac{3}{2}\int_0^2 \left(16\left(y^3 - 4y^2 + 4y\right) + \frac{27}{6}(y-2)^3\right) dy$$

$$= -\frac{3}{2}\left[16\left(\frac{y^4}{4} - \frac{4}{3}y^3 + 2y^2\right) + \frac{27}{24}(y-2)^4\right]_0^2$$

$$= -\frac{3}{2}\left(16\left(\frac{16}{4} - \frac{32}{3} + 8\right) - \frac{27 * 16}{24}\right)$$

$$= -5$$

The numerical solution is:

-5.0018 for the path integral and -5.0014 for the surface integral on a step size of 128.

### 7.1.2 Example 2

Lets take S to be the upper half surface of a sphere of radius 1 centered at the origin. The vector field is $\vec{F} = (2x - y, -yz^2, -y^2z)$.

The analytical solution is:

$$\vec{r} = \langle \cos t, \sin t, 0\rangle, 0 \le t \le 2\pi$$

$$d\vec{r} = \langle -\sin t, \cos t, 0\rangle dt$$

$$\vec{A}(t) = \langle 2\cos t - \sin t, 0, 0\rangle$$

$$\vec{A} \cdot d\vec{r} = \left(-2\cos t \sin t + \sin^2 t\right) dt$$

$$\oint_C \vec{A} \cdot d\vec{r} = \int_0^{2\pi} \left(-2\cos t \sin t + \sin^2 t\right) dt = \left[-2\frac{\sin^2 t}{2}\right]_0^{2\pi} + \frac{1}{2}\int_0^{2\pi} (1 - \cos 2t) dt$$

$$\oint_C \vec{A} \cdot d\vec{r} = 0 + \frac{1}{2}\left[t - \frac{\sin 2t}{2}\right]_0^{2\pi} = \pi$$

The numerical solution is:

3.1403 for the path integral and 3.1403 for the surface integral on a step size of 128.

## 7.2 Discussion

This system is a simple framework that can be extended to include more features and optimizations. The system can be extended to include more vector and scalar field operations and more visualization options. The system can also be optimized to handle larger data sets and more complex vector fields. A useful optimization would be to use a more accurate method to compute the partial derivatives such as symbolic differentiation. Another optimization would be to use a more efficient method to compute the divergence and curl of the vector field such as the finite element method[3] or the finite volume method[4]. A rigorous numerical analysis of the system would be useful to determine the accuracy and stability of the system. The system could also be extended to include more vector calculus theorems and operations such as the gradient theorem, the divergence theorem, and the Laplacian operator. A better tilling algorithm could be used to sample the surface at a more uniform intervals.

## 8 Conclusion

In this project, we have built a simple vector calculus framework to verify and visualize Stokes' curl theorem using basic numerical methods and the web graphics library three.js. We have implemented the integration of a scalar field over a path and a surface, the partial derivatives of a scalar field, the gradient of a scalar field, the divergence of a vector field, and the curl of a vector field. We have also visualized the vector field using arrows. We have verified the correctness of the implementation by comparing the results with the analytical solution. The system can be extended to include more features and optimizations and can be used to study more vector calculus theorems and operations.

## 9 References

## References

[1] M.H. El-Deeb. PEU-218 Assignments.

[2] M.H. El-Deeb. Stokes' Theorem (threejs).

[3] A. Harish. What is FEM and FEA explained: Finite Element Method.

[4] T Gallouët R. Eymard and R. Herbin. Finite Volume Methods.