# Приложение А ­ Диаграмма компонентов



# Приложение B ­Листинг программного кода

**Файл Surface.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

namespace SurfaceLibrary

{

public class Surface

{

protected double a { get; set; }

protected double b { get; set; }

protected double c { get; set; }

protected double d { get; set; }

public double XSt { get; set; }

public double XFin { get; set; }

public double YSt { get; set; }

public double YFin { get; set; }

public double Step { get; set; }

public Surface(double a, double b, double c, double d)

{

this.a = a;

this.b = b;

this.c = c;

this.d = d;

}

public void SetIntervals(double xSt, double xFin, double ySt, double yFin, double step)

{

XSt = xSt;

XFin = xFin;

YSt = ySt;

YFin = yFin;

Step = step;

}

public double GetSurfaceArea(Func<double, double, double, double, Func<double, double, double>, double> methodForCalculateIntegral, int maxCountThreads = 1000)

{

double x1 = XSt;

double x2 = x1 + Step;

int countThreads = 0;

List<Task<double>> tasks = new List<Task<double>>();

while (x1 < XFin)

{

var t = new Task<double>((state) =>

{

double localX1 = (state as (double, double)?).Value.Item1;

double localX2 = (state as (double, double)?).Value.Item2;

double s = 0;

double y1 = YSt;

double y2 = y1 + Step;

while (y1 < YFin)

{

localX2 = localX2 > XFin ? XFin : localX2;

y2 = y2 > YFin ? YFin : y2;

s += methodForCalculateIntegral(localX1, localX2, y1, y2, InitialFunction);

y1 += Step;

y2 += Step;

}

Interlocked.Decrement(ref countThreads);

return s;

}, (x1, x2));

// Simple spinlock

while (countThreads >= maxCountThreads)

{

Thread.Sleep(1);

}

countThreads++;

t.Start();

tasks.Add(t);

x1 += Step;

x2 += Step;

}

Task.WaitAll(tasks.ToArray());

return tasks.Select(x => x.Result).Sum();

}

/// <summary>

/// <para>Method for calculate area of surface</para>

/// <para>Return IEnumerable of Point. Point = { x, y, z }</para>

/// </summary>

public IEnumerable<Point> GetPoints()

{

double xTemp = XSt;

List<Task<List<Point>>> tasks = new List<Task<List<Point>>>();

while (xTemp <= XFin)

{

var t = new Task<List<Point>>((stateX) =>

{

double localX = (double)stateX;

double yTemp = YSt;

List<Point> localPoints = new List<Point>();

while (yTemp < YFin)

{

localPoints.Add(new Point { Z = DerivativeOfFuncion(localX, yTemp) });

yTemp += Step;

}

localPoints.Add(new Point { Z = DerivativeOfFuncion(localX, yTemp) });

return localPoints;

}, xTemp);

t.Start();

tasks.Add(t);

xTemp += Step;

}

Task.WaitAll(tasks.ToArray());

var points = tasks

.Select(t => t.Result)

.Aggregate((x, y) =>

{

x.AddRange(y.AsEnumerable());

return x;

})

.OrderBy(p => p.X)

.ThenBy(p => p.Y);

return points;

}

public virtual double InitialFunction(double x, double y)

{

return Math.Sqrt(1 +

Math.Pow(2 \* a \* x + c \* Math.Exp(-x), 2) +

Math.Pow(2 \* b \* y + d \* Math.Exp(-y), 2));

}

/// <summary>

/// Rule for calculate coordinate Z by X,Y.

/// </summary>

public virtual double DerivativeOfFuncion(double x, double y)

{

return a \* Math.Pow(x, 2) +

b \* Math.Pow(y, 2) +

c \* Math.Exp(-x) +

d \* Math.Exp(-y);

}

}

}

**Файл HomeController.cs:**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using Microsoft.AspNetCore.Mvc;

using CourseWork.Filters;

using SurfaceLibrary;

namespace CourseWork.Controllers

{

public class HomeController : Controller

{

Surface surface;

Func<double, double, double, double, Func<double, double, double>, double> IntegralCalculationMethod;

public HomeController(

Surface surface,

Func<double, double, double, double, Func<double, double, double>, double> IntegralCalculationMethod)

{

this.surface = surface;

this.IntegralCalculationMethod = IntegralCalculationMethod;

}

public IActionResult Index()

{

return View();

}

[PrepareFilter]

public IActionResult GetPoints(double xSt, double xFin, double ySt, double yFin, double step = 1)

{

surface.SetIntervals(xSt, xFin, ySt, yFin, step);

return Json(new

{

status = true,

result = surface.GetPoints()

});

}

[PrepareFilter]

public IActionResult GetSurfaceArea(double xSt, double xFin, double ySt, double yFin, double step = 1)

{

List<(int threadsCount, long time, double surfaceArea)> results = new List<(int, long, double)>();

Stopwatch stopwatch = new Stopwatch();

surface.SetIntervals(xSt, xFin, ySt, yFin, step);

for (int i = 1; i <= 10; i++)

{

stopwatch.Start();

double area = surface.GetSurfaceArea(IntegralCalculationMethod, i);

stopwatch.Stop();

results.Add((i, stopwatch.Elapsed.Milliseconds, area));

stopwatch.Reset();

}

return Json(new

{

status = true,

result = results.Select(json => new

{

json.threadsCount,

json.time,

json.surfaceArea

})

});

}

}

}

**Файл \_Layout.cshtml**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Surface Builder</title>

<**environment** **include**="Development">

<link rel="stylesheet" href="~/css/main.css" />

<link href="https://fonts.googleapis.com/css?family=Raleway" rel="stylesheet">

<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.1/css/all.css" integrity="sha384-50oBUHEmvpQ+1lW4y57PTFmhCaXp0ML5d60M1M7uH2+nqUivzIebhndOJK28anvf" crossorigin="anonymous">

<link rel="shortcut icon" type="image/png" href="~/favicon.ico">

</**environment**>

</head>

<body>

@RenderBody()

<**partial** **name**="\_CookieConsentPartial" />

<**environment** **include**="Development">

<**script** **src**="~/js/site.js" **asp-append-version**="true"></**script**>

<script src="~/js/view.js"></script>

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/js/plotly-latest.min.js"></script>

</**environment**>

<**environment** **exclude**="Development">

<**script** **src**="~/js/site.min.js" **asp-append-version**="true"></**script**>

<script src="~/js/plotly-latest.min.js"></script>

<script src="~/lib/jquery/dist/jquery.min.js"></script>

</**environment**>

<script>

let calcButton = document.getElementById('calc-button');

calcButton.addEventListener("click", \_ => {

if (errors.x === false && errors.y === false && errors.step === false) {

let data = getData();

graphicStub.style.visibility = 'visible';

barChartStub.style.visibility = 'visible';

document.getElementById('graphic-loading-span').innerHTML = 'Calculating...';

document.getElementById('bar-loading-span').innerHTML = 'Calculating...';

jump('surface');

new Promise((resolve, reject) => {

$.ajax({

url: '@Url.Action("GetPoints", "Home")',

data: {

xSt: data.xSt,

xFin: data.xFin,

ySt: data.ySt,

yFin: data.yFin,

step: data.step

},

success: response => {

if (response.status) {

let pointForPlot = distributePoints(response.result, data.xSt, data.xFin, data.ySt, data.yFin);

drawSurface(pointForPlot);

resolve();

}

else {

reject(response.msg);

}

},

error: \_ => reject()

});

}).then(\_ => {

$.ajax({

url: '@Url.Action("GetSurfaceArea", "Home")',

data: {

xSt: data.xSt,

xFin: data.xFin,

ySt: data.ySt,

yFin: data.yFin,

step: data.step

},

success: response => {

debugger;

if (response.status) {

drawBarChart(response.result);

document.querySelector('#area').innerHTML = response.result[0].surfaceArea;

document.querySelector('#fault').innerHTML = Math.abs(

response.result[0].surfaceArea / (8.222 \* 10 \*\* 30)

).toFixed(3);

}

else {

throw Exception(response.msg);

}

}

});

}).catch(err => console.log(err));

setTimeout(function () {

graphicStub.style.visibility = 'hidden';

barChartStub.style.visibility = 'hidden';

}, 2000);

} else {

graphicStub.style.visibility = 'visible';

barChartStub.style.visibility = 'visible';

document.getElementById('graphic-loading-span').innerHTML = 'Waiting for data entry...';

document.getElementById('bar-loading-span').innerHTML = 'Waiting for data entry...';

}

});

function distributePoints(points, xSt, xFin, ySt, yFin) {

let xArr = [];

let yArr = [];

let zArr = [];

for (let x = xSt; x <= xFin; x++) {

xArr.push(x);

}

for (let y = ySt; y <= yFin; y++) {

yArr.push(y);

}

for (let x = 0, index = 0; x < xArr.length; x++) {

let row = [];

for (let y = 0; y < yArr.length; y++, index++) {

row.push(points[index].z);

}

zArr.push(row);

}

return { x: xArr, y: yArr, z: zArr };

}

function drawSurface(points)

{

let x = points.x;

let y = points.y;

let z = points.z;

let data = [{

x: x,

y: y,

z: z,

type: 'surface',

"showscale": false,

"colorscale": [

[

0,

"#ff9800"

],

[

1,

"#ff9800"

]

],

"autocolorscale": false,

autosize: true

}];

let layout = {

font: {

size: 14,

color: 'rgb(24, 40, 64)',

family: 'Raleway'

},

autosize: true,

margin: {

l: 0,

r: 0,

t: 0,

b: 40

},

scene: {

camera: {

eye: {

x: 1.5,

y: 1.5,

z: 0.1

}

}

}

};

setTimeout(function () {

Plotly.newPlot('graphic', data, layout);

}, 1000)

}

function drawBarChart(results)

{

let threads = [];

let timings = [];

let areas = [];

for (let i = 0; i < results.length; i++) {

threads.push(results[i].threadsCount);

timings.push(results[i].time);

areas.push(results[i].surfaceArea);

}

debugger;

let data = [

{

x: threads,

y: timings,

text: timings,

textposition: 'auto',

type: 'bar',

mode: 'markers',

hoverinfo: 'y',

marker: {

color: '#ff9800',

line: {

width: 1,

color: '#444'

}

},

autosize: true

}

];

var layout = {

font: {

size: 18,

color: 'rgb(24, 40, 64)',

family: 'Raleway'

},

autosize: true,

margin: {

l: 80,

r: 40,

t: 40,

b: 40

},

xaxis: {

dtick: 1

},

yaxis: {

title: {

text: 'msec'

}

}

};

Plotly.newPlot('bar-chart', data, layout);

}

</script>

@RenderSection("Scripts", required: false)

</body>

</html>

**Файл SimpsonMethod.cs**

using System;

namespace SimpsonMethodLibrary

{

public class SimpsonMethod

{

public double Calculate(double xSt, double xFin, double ySt, double yFin, Func<double, double, double> f)

{

return (xFin - xSt) / 6 \* (yFin - ySt) / 6 \*

(f(xSt, ySt) + 4 \* f((xSt + xFin) / 2, ySt) +

f(xFin, ySt) + 4 \* (f(xSt, (ySt + yFin) / 2) + 4 \*

f((xSt + xFin) / 2, (ySt + yFin) / 2) + f(xFin, (ySt + yFin) / 2)) +

f(xSt, yFin) + 4 \* f((xSt + xFin) / 2, yFin) + f(xFin, yFin));

}

}

}