ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Předmět UPG

Dokumentace semestrální práce

Autor: Jakub Štrunc

Datum: 6.12.2024

Čas: kolem 160h

Obsah

Úvod

Cílem této práce bylo vytvořit interaktivní aplikaci pro simulaci a analýzu elektrostatického pole. Aplikace má uživateli umožnit vizualizaci vlastností pole generovaného bodovými náboji, včetně dynamické interakce s náboji, přidávání statických a dynamických sond a analýzy změn intenzity pole v čase. Hlavním mottem aplikace byla jednoduchá a rozšířená interakce se simulací. Uživatelé mohou snadno manipulovat s náboji a sondami, přidávat nové prvky, měnit scénáře a vizualizační režimy.

Stručný popis implementovaného řešení

Hlavní kód

Projekt je implementováno jako aplikace Windows Forms a hlavní logika je soustředěna do třídy DrawingPanel. Tato třída má na starosti jak výpočet šipek, tak jejich vykreslení, spolu s vykreslením nábojů a pohybující se sondy, která sleduje změny v elektrickém poli. Nejvýznamnější části řešení jsou:

- **DrawingPanel**: Třída zajišťující výpočet a vykreslení šipek, nábojů a sondy. Obsahuje metody pro inicializaci nábojů, generování šipek na základě Coulombova zákona a překreslení panelu.
- **Charge**: Třída reprezentující elektrický náboj s možností dynamického chování (např. pohyb po spline křivce nebo změna velikosti v čase) a zajišťuje jeho vykreslení v grafickém panelu, interakci s uživatelem pomocí formulářů a detekci kolizí s kurzorem myši.
- **Arrow**: Třída reprezentující šipky
- **Probe**: Třída reprezentující bodovou sondu, která může být statická nebo pohyblivá, měří intenzitu elektrického pole v daném bodě, umožňuje interakci uživatele prostřednictvím editoru vlastností a poskytuje vizualizaci jejího pohybu i hodnot elektrického pole v simulaci.
- **SideMenu**: Stará se o postranní nabídku, která poskytuje uživateli různé ovládací prvky a možnosti konfigurace, jako je výběr scénářů, správa objektů (náboje a sondy), ovládání režimů zobrazení, rychlosti simulace a další, přičemž umožňuje dynamické přizpůsobení rozložení při změně velikosti panelu.

- **Graph**: Třída `Graph` vytváří graf, který zobrazuje změny intenzity elektrického pole v čase pro vybrané sondy anebo pro nově vytvořenou sondu.
- **ShuntingYard**: Třída poskytuje metody pro převod matematických výrazů na postfixovou notaci (Reverzní Polská Notace - RPN) a jejich následné vyhodnocení pomocí zásobníku, podporující základní operátory (+, -,* , /, ^), matematické funkce (sin, cos, tan, exp) a proměnné jako t či konstanty pi a e.
- **Spline**: Třída SplineEditor umožňuje dynamické vytváření a úpravu Bézierových křivek s plynulou C^2 kontinuitou, přičemž poskytuje uživatelsky interaktivní funkce, jako je přidávání, přesouvání a mazání ovládacích bodů a jejich tangent, a zároveň umožňuje vizualizaci výsledné křivky na základě upravených dat.
- **DataCharged**: Třída ChargeData poskytuje strukturu pro ukládání dat o elektrickém náboji z JSON.
- **Script.json**: Soubor reprezentující náboje pro načítaní ze souboru

Legenda

Legenda zobrazuje škálu intenzity elektrického pole (v jednotkách N/C) v barevné mapě. Hodnoty na legendě reprezentují minimální, maximální a vybrané mezilehlé intenzity normalizované na barevné spektrum. Krajní body znamenají, že jsou nejmenší hodnota té barvi, neboť se to počítá ta barva nejdále od náboje.

Popis klíčových algoritmů

Vypočet elektrického pole

Funkce `CalculateElectricField` vypočítává celkové elektrické pole v určité pozici, způsobené všemi náboji v simulaci. Vektor elektrického pole je ovlivněn vzdáleností a hodnotou náboje každého jednotlivého náboje, a výsledkem je celkový vektor pole, který zohledňuje příspěvky všech nábojů. Vzoreček je modifikovaný, jelikož v zadání by šly šipky z minusu do plusu, což nedává smysl.

1. Inicializace celkového pole a iterace přes náboje

Funkce nejprve inicializuje vektor celkového pole **E_total** na nulu. Poté iteruje přes všechny náboje v poli náboje

```
E_total = Vector(0, 0)

foreach charge in charges:
```

2. Výpočet vzdálenosti a kontrola nulové vzdálenosti

Pro každý náboj se vypočítá vektor vzdálenosti **r** mezi jeho pozicí a zadaným bodem. Současně se vypočítá velikost této vzdálenosti **r**. Pokud je vzdálenost větší než nula, pokračuje se výpočtem.

```
r = position - charge.position

distance = sqrt(r.x^2 + r.y^2)

if distance > 0:
```

3. Normalizace vektoru a Coulombův zákon

Normalizuje se vektor vzdálenosti **r**, aby měl délku 1, a podle Coulombova zákona se vypočítá intenzita elektrického pole. Pole závisí na velikosti náboje a vzdálenosti **r**. Nakonec intenzitu elektrického pole vrátí.

```
r_normalized = Vector(r.x / distance, r.y / distance)

E_total += k * charge.magnitude * r_normalized / (distance^3)

return E_total
```

Vykreslení šipek

Šipky slouží k vizualizaci směru a velikosti elektrického pole v různých bodech prostoru. Každá šipka reprezentuje směr elektrického pole v určitém bodě a je vypočítána na základě konfigurace nábojů o které se stará funkce `GenerateArrows`.

1. Nastavení mřížky a iterace přes body

Nejprve se vypočítají zvětšené kroky mřížky a jejich poloviční hodnoty. Následně se stanoví hranice vykreslování na základě velikosti okna, posunu a měřítka. Je to připravené na implementaci nedodělaného rozšířeni. Poté metoda iteruje přes všechny body mřížky ve viditelné oblasti a pro každý bod volá funkci na vykreslení šipky.

```
scaledGridStepX = gridStepX * uniformScale

scaledGridStepY = gridStepY * uniformScale

halfStepX = scaledGridStepX / 2

halfStepY = scaledGridStepY / 2


maxX = (Width - panOffset.X + sideOff) / zoom
```

```
minX = (0 - panOffset.X - sideOff) / zoom

maxY = (Height - panOffset.Y + sideOff) / zoom

minY = (0 - panOffset.Y - sideOff) / zoom


for x in range(cx, maxX, scaledGridStepX):

    for y in range(cy, maxY, scaledGridStepY):

        DrawArrowAtGridPoint(g, cx, cy, x, y, halfStepX, halfStepY,
uniformScale)

    for y in range(cy - scaledGridStepY, minY, -scaledGridStepY):

        DrawArrowAtGridPoint(g, cx, cy, x, y, halfStepX, halfStepY,
uniformScale)
```

2. Vykreslení šipky

Šipka je vypočítána na pozici bodu, ale chceme mít její střed v tomto bodě. Musíme tedy šipku posunout o polovinu její délky. Pro každý bod se vypočítá vektor elektrického pole, jeho směr a velikost, a šipka se vykreslí na danou pozici.

```
position = Vector2(x + halfStepX - cx, y + halfStepY - cy)

E = ScaledCalculateElectricField(position)

if E.Length() > 0:

    dir = Normalize(E) * scaleFactor

    start = Vector2(x + halfStepX - dir.X / 2, y + halfStepY - dir.Y / 2)

    Arrow(start, dir).DrawArrow(g, arrowPen)
```

Vykreslení nábojů

Každý náboj je vizualizován jako kruh s barvou odpovídající jeho polaritě (červená pro kladné náboje a modrá pro záporné). Velikost kruhu je úměrná velikosti náboje, přičemž jeho hodnota je zobrazena ve středu kruhu.

1. Aktualizace vlastností částice

Metoda nejprve aktualizuje polaritu náboje a následně vypočítá její poloměr, který je určen na základě měřítka a velikosti polarity.

```
polarita = spočítáni Polish notation náboje na zaklade casu
radius = 15 * uniformScale + abs(Q) * 10 * uniformScale
```

2. Poloha náboje

Pokud je povoleno sledovaní trasy, metoda kontroluje platnost dráhy (Spline). Následně se vypočítá aktuální a budoucí pozice částice na dráze a tyto hodnoty se nastaví jako pozice částice.

```
if followPath:
    if Spline.invalid:
        showWarning("Spline is invalid")
        followPath = false
        return

    index = (time * 20) % Spline.path.Count
    xSpline = Spline.path[index].X
    ySpline = Spline.path[index].Y
    xSplineAhead = Spline.path[index + offset].X
    ySplineAhead = Spline.path[index + offset].Y
    Position = Vector2(xSplineAhead, ySplineAhead)
    x = cx + (xSpline * uniformScale) - radius
    y = cy + (ySpline * uniformScale) – radius
```

Pokud není aktivní pohyb podle dráhy, metoda vypočítá pozici částice na základě její statické polohy a vykresluje ji na základě aktuálního měřítka.

```
else:
    x = cx + (Position.X * uniformScale) - radius
    y = cy + (Position.Y * uniformScale) - radius
```

2. Vytvoření grafické reprezentace náboje:

Náboj je vykreslen jako kruh na vypočítané pozici s daným rádiusem. Barvy náboje jsou nastaveny takto: kladné náboje jsou červené, záporné modré. Kruh je vyplněn gradientem, který začíná barvou náboje ve středu a postupně přechází do bílé na okrajích pro hezčí efekt.

```
Vytvoř kruh na pozici předešlých výpočtů a vypočítaného poloměru

Barva = Červená, pokud polarita je pozitivní, jinak Modrá

Vyplň kruh gradientem (od barvy náboje do bílé)
```

3. Zobrazení hodnoty náboje:

Hodnota náboje (v coulombech) je zobrazena uprostřed kruhu. Font velikosti a stylu je přizpůsoben měřítku okna, aby byla hodnota dobře čitelná. Text je umístěn do středu kruhu tak, aby byl zarovnaný s pozicí náboje.

Vypočítej šířku a výšku textu

Text pozice X = `Position.X + Rádius náboje` – šířka textu / 2;

Text pozice Y = `Náboj pozice Y + Rádius náboje` – výška textu / 2;

Vykresli hodnotu náboje uprostřed kruhu

Pohyb sondy

Sonda vizualizuje směr a velikost elektrického pole v místě, kde se nachází. Pokud je sonda pohyblivá, pohybuje se po kruhové dráze, zatímco statická sonda zůstává na pevné pozici.

1. Vypočítání pozice sondy:

Statická sonda je vypočítána přímo na základě pevné souřadnice a měřítka.

```
if isStatic:

    Position = (cx + pointPosition.X * zoom, cy + pointPosition.Y * zoom)
```

Pohyblivá sonda se pohybuje po kruhové dráze. Pozice se vypočítává pomocí trigonometrických funkcí sinus a kosinus na základě aktuálního času, úhlové rychlosti a poloměru dráhy.

```
else:

    Position.X = cx + pointPosition.X + turnradius * cos(angularVelocity *
(time - startTime)) * uniformScale

    Position.Y = cy + pointPosition.Y + turnradius * sin(angularVelocity *
(time - startTime)) * uniformScale
```

2. Vypočítání elektrického pole a výkres šipky

Elektrické pole v aktuální pozici sondy je vypočítáno a vizualizováno jako šipka. Délka a směr šipky odpovídají intenzitě a směru elektrického pole.

```
position = (Position.X - cx, Position.Y - cy)

E = CalculateElectricField(position)

dir = Normalize(E) * 30 * uniformScale

arrow = Arrow(Position, dir)

arrow.DrawArrow(g, arrowPen)
```

3. Výkres sondy

Sonda je vykreslena jako kruh s definovaným poloměrem. Barva kruhu je gradientní, od středu (barva sondy) směrem k bílé, což zajišťuje estetický efekt.

```
probeRadius = radius * uniformScale

graphicsPath = createEllipse(Position.X - probeRadius, Position.Y -
probeRadius, 2 * probeRadius)

fillGradient(center=Color, outer=White)

g.fillEllipse(gradient)

g.drawEllipse(Border)
```

Barevná mapa

Barvy v barevné mapě odpovídají hodnotám intenzity pole, přičemž negativní hodnoty mají jiné odstíny než kladné.

1. Inicializace

Rozdělí se plocha na mřížku dle zvoleného kroku a připraví se dvourozměrné pole a seznam pro uchování hodnot intenzit.

```
step = 10 * uniformScale

widthSteps = Width / step

heightSteps = Height / step

fieldIntensities = new float[widthSteps + 1, heightSteps + 1]
```

```
intensityList = []
```

2. Výpočet intenzit pole

Pro každý bod mřížky je intenzita pole vypočítána jako součet příspěvků od všech nabojů. Každý příspěvek je úměrný velikosti náboje a nepřímo úměrný vzdálenosti.

```
for xStep in range(0, widthSteps):
    x = xStep * step
    for yStep in range(0, heightSteps):
        y = yStep * step
        intensity = 0
        for charge in charges:
            dx = (x - cx) - charge.Position.X * uniformScale
            dy = (y - cy) - charge.Position.Y * uniformScale
            r = sqrt(dx^2 + dy^2)
            if r > 1e-6:
                intensity += charge.Q / r
        intensity = min(intensity, 1e6)
        fieldIntensities[xStep, yStep] = intensity
        intensityList.append(intensity)
```

3. Výpočet mezí intenzity

Seznam intenzit je seřazen a určují se kvantilové hodnoty pro dolní a horní meze. Tyto hodnoty se použijí pro normalizaci barev.

```
intensityList.sort()
minIntensity = intensityList[5%]
maxIntensity = intensityList[95%]
minlevelIntensity = intensityList[1%]
maxlevelIntensity = intensityList[99%]
maxAbsIntensity = max(abs(minIntensity), abs(maxIntensity))
if maxAbsIntensity == 0:
    maxAbsIntensity = 1e-5
```

4. Generování barevné mapy

Iterujeme přes mřížku a vykreslí barevné čtverce pro každou buňku. Barva každého čtverce je určena normalizovanou intenzitou pole.

```
if showColorMap:
    for xStep in range(0, widthSteps):
        x = cx + (xStep - widthSteps / 2 - 1) * step
        for yStep in range(0, heightSteps):
            y = cy + (yStep - heightSteps / 2 - 1) * step
            intensity = fieldIntensities[xStep, yStep]
            normalizedIntensity = intensity / maxAbsIntensity
            normalizedIntensity = clamp(normalizedIntensity, -1, 1)
            brush.Color = NegativeMapIntensityToColor(normalizedIntensity) if
normalizedIntensity < 0 else MapIntensityToColor(normalizedIntensity)
            g.FillRectangle(brush, x, y, step + 1, step + 1)
```

Siločáry

Metoda `DrawFieldLines` vykresluje siločáry elektrického pole pro každou částici (charge) v dané oblasti. Siločáry vizualizují směr a intenzitu elektrického pole, přičemž zahrnují šipky indikující směr pole.

1. Inicializace parametrů siločar

Pro každou částici se inicializují parametry nutné pro vykreslování siločar. Počet siločar je pevně nastaven na osm a délka jednoho kroku siločáry je určena konstantní hodnotou.

```
for charge in charges:
    numberOfLines = 8
    stepLength = 5
    minDistanceSquared = (charge.radius * uniformScale * 0.1)^2
```

2. Vykreslení siločar

Každá siločára začíná na obvodu částice, přičemž její počáteční bod je určen úhlem odpovídajícím pořadí siločáry. Poté se iterativně prodlužuje v závislosti na směru elektrického pole vypočítaném v aktuálním bodě. Siločára se prodlužuje buď směrem k pozitivním nábojům, nebo směrem od negativních nábojů.

```
for i from 0 to numberOfLines:
    angle = i * 2 * PI / numberOfLines
    startPoint = (charge.Position.X + radius * cos(angle), charge.Position.Y + radius * sin(angle))

    while steps < 1000:
        electricField = ScaledCalculateElectricField(startPoint - (cx, cy))
        if electricField.LengthSquared == 0:
            break

        direction = Normalize(electricField) * stepLength
        endPoint = startPoint + direction if charge.Q > 0 else startPoint - direction
```

3. Kontrola vzdáleností

Během každého kroku se kontroluje vzdálenost mezi aktuálním koncovým bodem siločáry a částicí. Pokud se siločára přiblíží k částici na vzdálenost menší než definované minimum, iterace se ukončí.

```
if DistanceSquared(endPoint, charge.Position + (cx, cy)) < minDistanceSquared:
    break
```

4. Vykreslení úseček a šipek

Každý krok siločáry je vykreslen jako úsečka spojující aktuální počáteční a koncový bod. Pro vizualizaci směru elektrického pole jsou na každé desáté úsečce vykresleny šipky.

```
DrawLine(g, startPoint, endPoint)
if steps % 10 == 0 or steps == 999:
    DrawArrow(g, pen, startPoint, endPoint, charge.Q > 0)
```

Vrstevnice

Vrstevnice jsou čáry spojující body se stejnou intenzitou, což umožňuje vizualizaci gradientů a změn v poli na ploše.

1. Inicializace vrstevnic

Vypočítá se adaptivní počet vrstevnic pomocí metody `CalculateAdaptiveContourLevels`, která bere v úvahu rozsah intenzit a maximální povolený počet vrstevnic. Interval mezi vrstevnicemi je určen rozdílem mezi minimální a maximální intenzitou děleným počtem vrstevnic

```
maxContourLevels = 20

    contourLevels = CalculateAdaptiveContourLevels(minlevelIntensity, maxlevelIntensity, maxContourLevels)

    contourInterval = (maxlevelIntensity - minlevelIntensity) / contourLevels
```

2. Vykreslení vrstevnic

Iterujeme přes buňky mřížky a hledá průsečíky vrstevnic s hranami buněk. Pokud jsou nalezeny dva průsečíky, vykreslí se mezi nimi úsečka.

```
for xStep from 0 to widthSteps - 1:
        x = xStep * step
        for yStep from 0 to heightSteps - 1:
            y = yStep * step

            bl = fieldIntensities[xStep, yStep]
            br = fieldIntensities[xStep+1, yStep]
            tl = fieldIntensities[xStep, yStep+1]
            tr = fieldIntensities[xStep+1, yStep+1]

            points = [] // List of intersection points

            if intersects(bl, tl, contourLevel):
                t = (contourLevel - bl) / (tl - bl)
```

```
                points.append(Point(x, y + step * t))


            if intersects(br, tr, contourLevel):
                t = (contourLevel - br) / (tr - br)
                points.append(Point(x + step, y + step * t))


            if intersects(bl, br, contourLevel):
                t = (contourLevel - bl) / (br - bl)
                points.append(Point(x + step * t, y))


            if intersects(tl, tr, contourLevel):
                t = (contourLevel - tl) / (tr - tl)
                points.append(Point(x + step * t, y + step))


            if points.length == 2:
                drawLine(g, points[0], points[1], uniformScale)
```

Shunting Yard

Metoda `ConvertToPostfix` převádí matematický výraz z infixového zápisu (např. 3 + 4 * 2) na Reverse Polish Notation (např. 3 4 2 * +). Tento formát je vhodný pro snadné vyhodnocení matematických výrazů bez potřeby závorek, protože pořadí operací je implicitně dáno pořadím tokenů.

1. Inicializace datových struktur

- **Seznam postfix**, kam se postupně ukládají výsledné tokeny ve formátu postfix.
- **Zásobník operators**, který slouží pro dočasné uložení operátorů, závorek a funkcí během zpracování.

2. Zpracování jednotlivých tokenů

Každý token je zpracováván podle svého typu:

a) Čísla a proměnné

Pokud je token číslo (např. 3.14), proměnná (t) nebo konstanta (pi), je ihned přidán do seznamu postfix.

b) Otevírací závorky

Otevírací závorka je uložena do zásobníku pro pozdější zpracování.

c) Uzavírací závorky

Při nalezení uzavírací závorky se všechny operátory ze zásobníku přesouvají do seznamu postfix, dokud se nenarazí na otevírací závorku. Tato otevírací závorka je poté ze zásobníku odstraněna.

d) Operátory (+, -, *, /, atd.)

Při zpracování operátorů se kontroluje jejich priorita. Operátory na vrcholu zásobníku s vyšší nebo stejnou prioritou jsou přesunuty do seznamu postfix. Poté se aktuální operátor uloží do zásobníku.

Operátor	Priorita
+	1
-	1
*	2
/	2
^	3

e) Funkce (např. sin, cos)

Funkce jsou přímo vloženy do zásobníku a zpracovány, když se objeví jejich argumenty.

Spliny

Skládá se z několika kroků, které zahrnují vykreslení bodů, tečen, křivky a celkovou vizualizaci.

1. Vykreslení řídicích bodů a tečen

Řídicí body jsou transformovány podle aktuálního měřítka a posunutí a vykresleny jako modré kruhy. Tečny jsou vykresleny jako červené kruhy na odpovídajících pozicích.

```
foreach point in ControlPoints:
    adjustedX = point.X * drawingPanel.uniformScale + drawingPanel.cx
    adjustedY = point.Y * drawingPanel.uniformScale + drawingPanel.cy
    draw blue circle at (adjustedX, adjustedY)

foreach tangent in Tangents:
    adjustedX = tangent.X * drawingPanel.uniformScale + drawingPanel.cx
    adjustedY = tangent.Y * drawingPanel.uniformScale + drawingPanel.cy
```

```
    draw red circle at (adjustedX, adjustedY)

    if tangent is middle:
        connect to adjacent control points with gray lines
    else if tangent is endpoint:
        connect to nearest control point with gray line
    adjustedX = tangent.X * drawingPanel.uniformScale + drawingPanel.cx
    adjustedY = tangent.Y * drawingPanel.uniformScale + drawingPanel.cy
    draw red circle at (adjustedX, adjustedY)

    if tangent is middle:
        connect to adjacent control points with gray lines
    else if tangent is endpoint:
        connect to nearest control point with gray line
```

2. Výpočet bodů na Bézierově křivce

Bod na křivce je vypočítán pomocí kombinace čtyř řídicích bodů a parametru t určujícího polohu na intervalu [0, 1]. Viz. Zdroj: https://en.wikipedia.org/wiki/B%C3%A9zier_curve

```
u = 1 - t
tt = t * t
uu = u * u
uuu = uu * u
ttt = tt * t

x = (uuu * p0.X) + (3 * uu * t * p1.X) + (3 * u * tt * p2.X) + (ttt * p3.X)
y = (uuu * p0.Y) + (3 * uu * t * p1.Y) + (3 * u * tt * p2.Y) + (ttt * p3.Y)
```

3. Vykreslení řídicích bodů a tečen

Křivka je vykreslena iterativním výpočtem bodů na základě parametrické rovnice Bézierovy křivky. Tyto body jsou transformovány a spojeny do hladké čáry.

```
if ControlPoints.Count < 2:
    return
```

```
foreach control point segment:
    calculate cubic Bézier curve points between p0, p1, p2, p3
    add points to path if distance to last point >= segmentLength

transform path points to screen coordinates
connect all points with a line
```

Graf

1. Inicializace grafu

Proces inicializace a aktualizace grafu zajišťuje vytvoření grafického okna, konfiguraci os a datových sérií, následně pak pravidelné přidávání aktuálních hodnot intenzity elektrického pole.

2. Aktualizace grafu

Aktualizace grafu probíhá v asynchronním cyklu, který v pravidelných intervalech získává aktuální hodnoty intenzity elektrické. Tento kód pravidelně aktualizuje graf tím, že přidává aktuální hodnoty intenzity elektrického pole ze sond do odpovídajících datových sérií, přičemž zajišťuje bezpečné zpracování operací v hlavním vlákně aplikace.

```
while true:
  if (okno se zavrelo):
    break
  elapsedTime = currentTime - startTime
  xAxis.Labels = range(0, number of points in first series)

  wait 1 second
```

Omezení a zjednodušení

Velikost okna

Řešení aplikace vyžaduje minimální velikost grafického panelu 800×600 pixelů, aby byla zajištěna správná čitelnost a přehlednost všech vizualizovaných prvků. Tato velikost byla zvolena tak, aby poskytla dostatečný prostor pro zobrazení všech klíčových komponent simulace a zároveň předešla problémům, které by mohly vzniknout při menším rozlišení:

- **Kvalita barevné mapy:** Barevná mapa, která znázorňuje intenzitu elektrického pole v prostoru, by při menších rozměrech okna mohla ztratit detailnost a plynulost přechodů barev. Menší velikost by mohla způsobit, že jemné nuance v intenzitách pole by nebyly správně viditelné, což by omezilo její interpretovatelnost.
- **Čitelnost šipek:** Šipky reprezentující směr a velikost elektrického pole by mohly být při menším okně příliš malé nebo dokonce neviditelné. Jelikož šipky hrají klíčovou roli při znázorňování směru pole, jejich čitelnost je zásadní pro pochopení simulace. Nedostatečně čitelné šipky by mohly způsobit zmatení uživatele a snížení přínosu vizualizace.
- **Zobrazení prvků:** Grafická reprezentace objektů pomocí kruhů je navržena tak, aby jasně odrážela jejich velikost. Při menším rozlišení by kruhy mohly být příliš malé, což by mohlo vést k problémům s jejich rozlišitelností. Textová informace uvnitř kruhu, která udává hodnotu náboje, by také mohla být nečitelná, což by dále zhoršilo pochopení situace.
- **Rozmístění prvků:** Menší okno by mohlo vést k přetížení zobrazené scény, kde by jednotlivé prvky (náboje, šipky, sondy) byly příliš blízko u sebe. To by mohlo způsobit vizuální nepřehlednost a znemožnit přesné rozlišení jejich jednotlivých funkcí.

Krokování v barevné mapě

Pro zajištění plynulého chodu programu a optimalizaci výkonu není mapa vykreslována po jednotlivých pixelech, ale je vzorkována v pevných krocích o velikosti 10 pixelů. Tento přístup výrazně snižuje výpočetní náročnost, protože intenzita pole se vypočítává pouze na bodech s pevnou roztečí, nikoliv na každém jednotlivém pixelu. Výsledkem je plynulejší provoz aplikace, i při aktivaci dalších vizualizačních prvků, jako jsou šipky intenzity nebo více sond.

Instalace a spuštění aplikace

Aplikace je implementována v jazyce C# jako Windows Forms aplikace. Pro spuštění projektu postupujte následovně:

1. **Závislosti:** Visual Studio 2022 nebo novější, .NET Framework 4.7.2 nebo novější.
2. **Build:** Kliknutí na build.
3. **Spuštění:** Otevření příkazové řádky v hlavním adresáři projektu a zadání příkazu `Run <script> -g<X>x<Y>, kde <script>` je číslo od 0 do 4 reprezentující scénář a `<X>` a `<Y>` jsou celočíselné hodnoty udávající rozteč vzorků mřížky v osách x a y v pixelech. Tyto údaje jsou volitelné a nemusí být zadány.

Ovládání aplikace

Přidávání sond (pro splnění zadaní)

Uživatel může přidávat statické sondy do simulace pomocí kombinace **Shift + kliknutí** myší na libovolné místo v prostoru simulace. Po přidání sondy se automaticky zobrazí graf, který sleduje průběh intenzity elektrického pole v místě sondy v závislosti na čase. Graf umožňuje vizualizovat změny v poli způsobené časově proměnnými náboji nebo jejich pohybem.

Posouvání nábojů a sond

Bodové náboje a sondy lze libovolně posouvat myší na jiná místa simulace. Stačí kliknout a přetáhnout vybraný objekt na požadovanou pozici. Při posouvání je vizualizace pole okamžitě aktualizována

Menu Button

Na pravé straně aplikace se nachází **menu tlačítko**, které po kliknutí rozbalí postranní nabídku. Tato nabídka obsahuje intuitivní možnosti pro interakci s aplikací, jako je přepínání scénářů, přidávání objektů (nábojů a sond), změna režimů vizualizace, úprava rychlosti simulace a další.:

Udělané rozšíření a Jejich ovládání

Načtení scénáře ze souboru

Aplikace umožňuje uživatelům v menu v okně **Scenarios** vybrat JSON soubor, který definuje vlastní konfiguraci elektrických nábojů. Tento JSON soubor musí mít specifickou strukturu, která odpovídá následujícímu formátu:

```
[
  {
    "Q": "sin ( t )",
    "PositionX": -1,
    "PositionY": 0
  },
  {
    "Q": "-1",
    "PositionX": 1,
    "PositionY": 0
```

```
  }
]
```

Kde objekt reprezentuje naboj s vlastnostmi:

Q: Hodnota náboje je zadaná ve formě výrazu nebo konstanty. Výraz může být závislý na čase (t), což umožňuje časově proměnné náboje (např. "sin (t)"). Každý parametr výrazu musí být oddělen mezerou (například "1*cos(t)" musí být zapsáno jako "1 * cos (t)").

PositionX: X-ová souřadnice náboje ve dvourozměrném prostoru.

PositionY: Y-ová souřadnice náboje ve dvourozměrném prostoru.

Editor statických scénářů

Uživatel má v menu, v okně **Objects**, možnost přidávat bodové náboje a sondy, které jsou po vytvoření automaticky umístěny do středu okna. Tyto objekty lze myší přesouvat na libovolné místo scény, nebo na ně dvojitě kliknout, čímž se otevře okno pro jejich podrobné úpravy. V tomto okně lze změnit parametry, jako je pozice a velikost náboje, vlastnosti sondy nebo jejich vymazaní.

Editor dynamických scénářů

Editor dynamických scénářů umožňuje uživateli v aplikaci přidávat, upravovat a odstraňovat bodové náboje s časově proměnnou velikostí. Pokud uživatel na **náboj dvojklikne**, zobrazí se okno, ve kterém může v textovém poli **Q** zadat výraz reprezentující velikost náboje. Podporovány jsou základní funkce jako sin x, cos x, tan x, e^-x, pi a výrazy mohou být závislé na čase (t), což umožňuje simulovat časově proměnné chování. Každý parametr výrazu musí být oddělen mezerou (například "1*cos(t)" musí být zapsáno jako "1 * cos (t)"). Aplikace nejen umožňuje editaci těchto dynamických scénářů, ale i jejich přehrání, což uživatelům poskytuje přehled o chování elektrického pole v průběhu času.

PS. Toto se celkem dobře sešlo, a ne jenom že mám bonusové body ale i základ semestrální práci z C. :)

Náboje pohybující se v čase

Aplikace je rozšířena o možnost pohybu náboje po kubických křivkách, které jsou vytvořeny pomocí Bézierových spline. Uživatel může v menu, v okně **Path**, definovat a upravovat trajektorie pohybu. Pokud je path červená znamená to, že je nevalidní a náboj ji nebude sledovat. **Dvojklikem levým tlačítkem** na bod v cestě může přidat nový bod mezi dva existující, zatímco **kliknutím pravým tlačítkem** může vybraný bod odstranit.

Cesty jsou implementovány jako spojení několika Bézierových křivek s hladkou kontinuitou. Pro přiřazení pohybu náboje k určité trajektorii musí uživatel **dvojkliknout na daný náboj** a v otevřeném okně zaškrtnout možnost **Follow Path**. Toto se musí udělat po nastavení křivky, jinak n8boj se pochybovat nebude. Náboj se následně začne pohybovat po zvolené křivce, přičemž jeho pozice je automaticky aktualizována v závislosti na čase.

Zrychlení/zpomalení simulace

V menu, v okně **Speed**, má uživatel možnost zvolit rychlost simulace pomocí předdefinovaných voleb, jako je 2× zrychlení, 2× zpomalení nebo návrat do původní rychlosti, což zahrnuje pohyb sondy, pohyb nábojů a změny velikosti nábojů.

Vrstevnice

Vrstevnice v aplikaci jsou vytvořeny jako izočáry, které znázorňují oblasti s konstantní intenzitou elektrického pole. Samotné vykreslení vrstevnic probíhá interpolací mezi body mřížky, kde intenzita překračuje předem definované hladiny (konturové hodnoty). Pro každý čtyřúhelník na mřížce se určí průsečíky těchto hladin s hranami čtyřúhelníku a tyto body jsou spojeny úsečkami, které tvoří izočáry.

Vrstevnice lze snadno zapnout v menu prostřednictvím okna **Modes** zaškrtnutím možnosti **Field Lines**. Tato možnost umožňuje uživateli lépe pochopit rozložení intenzity elektrického pole a její gradienty, což doplňuje další vizualizační prvky, jako je barevná mapa nebo šipky intenzity.

Siločáry

Siločáry znázorňují směr elektrického pole a jsou generovány jako křivky, které začínají v blízkosti nábojů a sledují směr pole v každém bodě prostoru. Lze je zapnout v menu prostřednictvím okna **Modes** zaškrtnutím možnosti **Level Lines.** Proces výpočtu začíná umístěním počátečních bodů na povrch náboje, odkud vychází siločáry (pro kladné náboje směrem ven, pro záporné směrem dovnitř). Poté se v každém kroku počítá elektrické pole v aktuální pozici na základě Coulombova zákona a na základě směru pole se posouvá bod siločáry. Pohyb se zastaví, pokud bod dosáhne jiné oblasti s vysokou intenzitou, dostane se příliš blízko k jinému náboji, nebo pokud počet kroků přesáhne limit.

Více sond

Aplikace umožňuje uživateli přidávat více sond a zobrazovat jejich data v grafu. Sondy lze přidat do grafu dvojklikem na konkrétní sondu, čímž se otevře dialogové okno, kde je třeba zaškrtnout pole **Graph**. Pro zobrazit graf musí uživatel výběrem možnosti **Graph** v menu. Graf zobrazuje přehledný průběh intenzity elektrického pole pro všechny vybrané sondy, přičemž každá sonda má svou vlastní barevně odlišenou křivku.

Pokud uživatel graf otevře, aniž by předtím přidal do něj jakoukoliv sondu, aplikace zobrazí varovné okno s informací, že v grafu nejsou žádná data k zobrazení.

Závěr

Semestrální práce byla úspěšně dokončena a aplikace splnila většinu stanovených cílů. I přes několik drobných chyb a bugů se podařilo vytvořit funkční a interaktivní nástroj, který dokáže efektivně simulovat elektrostatické pole a umožňuje jeho analýzu. Velké množství

implementovaných funkcí odráží moje zaujetí tématem a radost z práce na projektu, která vedla k tomu, že jsem projektu věnoval více času, než bylo původně plánováno. 😊