Звіт
з обчислюваної геометрії
та комп’ютерної крафіки

Ганевича Вадіма

09.02.2022

Умова завдання:

30. Дано два трикутники, які не перетинаються. В кожному з цих трикутників вибирається по одній точці. Перший трикутник починає рухатись до другого паралельно вектору, що з’єднує ці точки. Зупинити рух трикутника в момент його дотику до другого трикутника. Узагальнити задачу для довільного полігона.

Геометричний розв’язок:

Крок 1. (Задання координат полігона)

Для задання полігону використаємо формули для знаходження координат правильних многокутників:

$$x_i = x_0 + R\cos\left(\phi_0 + \frac{2\pi i}{n}\right),$$
$$y_i = y_0 + R\sin\left(\phi_0 + \frac{2\pi i}{n}\right),$$

Для загальних випадків візьмемо радіус в 100px.
Для випадків, коли центр полігону розпашований близько до краю екрана (менше 100px) - знайдемо найменшу відстань до краю екрана (позн. X) та визначимо радіус як X - 5px.

Крок 2. (Рух полігона)

Зобразимо другий полігон на головному шарі екрану, цей шар буде статичним та не піддаватиметься змінам.

Перший полігон на вектор руху (відрізок, що з’єднує центри полігонів) зобразимо на іншому шарі, який буде очищуватись та перемальовуватись.

Для руху по вектору потрібно розділити координати цього вектора на довільне n та поступово додавати до координат полігона, що рухається.
У моєму випадку n = 1000.

Крок 3. (Зупинка руху)

Для того, щоб перевірити, чи задані полігони не дотикаються одне до одного я використав наступний алгоритм:

1) Вибираю дві сусідні вершини одного з полігонів
2) Знаходжу довжину сторони, яку утворюють ці вершини
3) Випускаю вектори з цих вершин до довільної вершини другого полігона
4) Шукаю суму довжин цих векторів
5) Порівнюю цю суму з раніше знайденою довжиною сторони
6) Якщо модуль різниці менше 1e-3 (0,00x), то вершина, до якої направлені вектори, дотикається до сторони, утвореної даними двома точками
7) Повторю алгоритм для кожної вершини другого полігона, поки не знайду вершину, що задовільняє умову 6
8) Повторюю алгорит для кожної пари сусідніх точок першого полігона, поки не виконається умова 6
9) Міняю полігони місцями, та проводжу ще одну таку ж перевірку

Якщо знайдено дотик - зупиняю рух, якщо дотику немає - рух продовжується.

Реалізація в коді:

Крок 1.

Для зображення полігона нам потрібна точка та кількість сторін. Щоб програма не була надто статичною - дозволимо користувачу вводити цю інформацію.

Використаю наступну функцію для контролю введення інформацію від користувача та парсингу її на масив чисел:

```
// функція парсить та перевіряє на коректність вхідні данні від користувача
function input(){
    let checkLen = (tmpString) => {...}

    let parse = (string) => {...}

    let tmp = null;

    while (true) {
        if (tmp !== null) {
            tmp = tmp.split(' ');
            if (checkLen(tmp)){
                return parse(tmp);
            } else {
                tmp = null;
            }
        }
        else {
            tmp = window.prompt(
                message: 'Введіть, будь ласка, інформацію про полігон: кількість сторін та координати точки, ' +
                'навколо якої буде побудований полігон! ' +
                '\n У вас наступне розширення екрану: ' + window.screen.width + 'x' + window.screen.height);
        }
    }
}
```

Коли ми отримали потрібну інформацію від користувача - потрібно знайти координати вершин полігона:

```
// вираховує координати вершин полігона
#getCoordinate () {
    let points = [];
    let radius = this.#getRadius();
    let angle = 360;
    let koefOfAngle = 10;

    points[0] = {
        x: this.mainPoint.x + radius,
        y: this.mainPoint.y
    }

    for (let i = 0; i < this.numberOfLines; i++){
        points[i] = {
            x: this.mainPoint.x + radius * Math.cos( x: koefOfAngle + (2 * Math.PI * i) / this.numberOfLines),
            y: this.mainPoint.y + radius * Math.sin( x: koefOfAngle + (2 * Math.PI * i) / this.numberOfLines)
        }
    }

    return points;
}
```

Для формул нам потрібно знайти радус, що й відбувається на 4 радку попереднього скріншота. Функція знаходження радіуса:

```
// вираховує максимально допустимий радіус полігона
#getRadius () {
    let maxRadius = 105;

    if (this.mainPoint.x <= 100 || this.mainPoint.x >= window.screen.width - 100){
        maxRadius = Math.min(this.mainPoint.x, window.screen.width - this.mainPoint.x);
    }

    if (this.mainPoint.y <= 100 || this.mainPoint.y >= window.screen.height - 100){
        maxRadius = Math.min(this.mainPoint.y, window.screen.height - this.mainPoint.y);
    }

    return maxRadius - 5;
}
}
```

Крок 2.

З’єднуючи знайдені координати вершин отримаємо правильний многокутник. Для виведення його на екран використаємо наступну функцію:

```
// намалювати полігони та з'єднати їх центри
show () {
    this.#showPolygon(this.polygon1, this.layer);
    this.#showPolygon(this.polygon2, this.stage);

    this.layer.path()
        .moveTo(this.polygon1.mainPoint.x, this.polygon1.mainPoint.y)
        .lineTo(this.polygon2.mainPoint.x, this.polygon2.mainPoint.y);
}
```

Ця функція з'єднує центри двох полігонів та викликає функцію, котра малює окремий полігон на заданою шарі:

```
// намалювати полігон на певному фреймі
#showPolygon (polygon, layer){
    for (let i = 1; i < polygon.points.length; i++){
        layer.path()
            .moveTo(polygon.points[i-1].x, polygon.points[i-1].y)
            .lineTo(polygon.points[i].x, polygon.points[i].y);
    }
    layer.path()
        .moveTo(polygon.points[polygon.points.length - 1].x, polygon.points[polygon.points.length - 1].y)
        .lineTo(polygon.points[0].x, polygon.points[0].y);
}
```

Визначмо змінну, що зберігатиме крок переміщення для полігона:

```
this.step = {
    x: (this.polygon2.mainPoint.x - this.polygon1.mainPoint.x) / 1000,
    y: (this.polygon2.mainPoint.y - this.polygon1.mainPoint.y) / 1000
}
```

Та, за допомогою наступного цикла змусимо перший полігон рухатись до другого:

```
// рухати перший полігон до другого
async move() {

    while (!this.#check(this.polygon1, this.polygon2) && !this.#check(this.polygon2, this.polygon1)) {
        this.#clear();
        this.#updatePoints();
        this.#showPolygon(this.polygon1, this.layer)
        this.layer.path()
            .moveTo(this.polygon1.mainPoint.x, this.polygon1.mainPoint.y)
            .lineTo(this.polygon2.mainPoint.x, this.polygon2.mainPoint.y);


        // sleep
        await new Promise( executor: resolve => setTimeout(resolve,  timeout: 10));

    }
}
```

Цей цикл можно розбити на такі кроки:

1) Шар, на якому зображений перший полігон очищується
2) Координати першого полігона обновляються
3) Полігон малюється заново по обновлених координатах

Крок 3.

Тепер звернемо увагу на умову, що знаходиться в циклі while. Розглянемо цю функцію:

```
// перевірка на дотик двох полігонів
#check (polygon1 :Polygon = new Polygon(), polygon2 :Polygon = new Polygon()) {
    for (let i = 1; i <= polygon1.numberOfLines; i++){
        let x1 = polygon1.points[i-1].x;
        let y1 = polygon1.points[i-1].y;
        let x2, y2;
        if (i === polygon1.numberOfLines){
            x2 = polygon1.points[0].x;
            y2 = polygon1.points[0].y;
        } else {
            x2 = polygon1.points[i].x;
            y2 = polygon1.points[i].y;
        }

        let len = Math.sqrt( x: (x2 - x1)**2 + (y2 - y1)**2);

        for (let k = 0; k < polygon2.numberOfLines; k++){
            let x3 = polygon2.points[k].x;
            let y3 = polygon2.points[k].y;

            let distance = Math.sqrt( x: (x3 - x1)**2 + (y3 - y1)**2);
            distance += Math.sqrt( x: (x3 - x2)**2 + (y3 - y2)**2);
            if (Math.abs( x: len - distance) < 1e-2) return true;
        }
    }
    return false;
}
```

Для скорочення коду вводимо змінні x1, x2…
Шукаємо довжину між двома сусідніми точками та зберігаємо її в змінній len.
Для кожної вершини іншого полігона шукаємо суму векторів, що направлені від раніше зафіксованих точок до цієї вершини, та зберігаємо це значення в змінній distance.
Порівнюємо значення distance з len та повертаємо true у випадку, коли умова порівняння виконується.

Повторюмо цей алгоритм допоки не виконається умова, або ми не розгянемо всеможливі комбінації: сторона - вершина.
Якщо умова так і не виконалась - повертаємо false.

Приклади роботи програми: