Grafika komputerowa
Projekt 2.
Eliminacja elementów zasłoniętych

Radosław Churski

13 listopada 2018

1 Fundamentalne zmiany względem projektu pierwszego

W projekcie pierwszym operowałem odcinkami. Obiekty reprezentowałem jako odcinki, translacje czy obroty były wykonywane na nich i samo wyświetlanie obiektów po rzutowaniu również realizowane było na odcinkach.

Aby móc analizować to, które powierzchnie będą zasłonięte, a które będą zasłaniać musiałem zadbać o to, aby obiekty były reprezentowane przez ich ściany, na nich wykonywać wszelkie operacje i je wyświetlać po rzutowaniu.

2 Eliminacja elementów zasłoniętych

2.1 Dostosowanie sposobu wyświetlania

Wstępnym krokiem do zastosowalna eliminacji elementów zasłoniętych, była zmiana sposoby wyświetlania obiektów - teraz wyświetlane ściany powinny być wypełnione. Obecnie funkcja odpowiadająca za rysownie obiektów ma następującą postać:

```
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;
    g2d.setStroke(new BasicStroke(2));
    for (Polygon polygon: this.polygonsToView) {
        g2d.setColor(Color.BLACK);
        g2d.draw(polygon);
        g2d.setColor(Color.GREEN);
        g2d.fillPolygon(polygon);
    }
}
```

Dla każdego wielokąta rysujemy na czarno jego obwód a następnie na zielono jego wypełnienie.

Dzięki tej zmianie wystarczy zmienić kolejność zapisanych ścian, aby zastosować algorytm malarski.

2.2 Algorytm malarski

Algorytm malarski jest jednym z najprostszych algorytmów eliminacji elementów zasłoniętych. Polega on na nakładaniu kolejnych warstw obrazu, zaczynając od najdalszej i kończąc na najbliższej. Problematyczne w tym algorytmie jest posortowanie obiektów, które będą rysowane.

Algorytm malarski działa dla bardzo prostych modeli. Wystarczy, żeby obiekty się przecinały, aby nie móc go zastosować.

2.3 Sortowanie

W swoim rozwiązaniu zastosowałem trzy rodzaje sortowania względem położenia kamery:

2.3.1 Sortowanie względem środka ciężkości

Środek ciężkości postanowiłem wyznaczać jako średnią arytmetyczną współrzędnych wszystkich wierzchołków.

```
private double getMiddleDistance(Point point) {
    double avgX = 0;
    double avgY = 0;
    double avgZ = 0;
    for (Point vertex: this.vertices) {
        avgX += vertex.getX();
        avgY += vertex.getY();
        avgZ += vertex.getZ();
    }
    avgX /= this.vertices.size();
    avgY /= this.vertices.size();
    avgZ /= this.vertices.size();
    return point.getDistance(new Point(avgX, avgY, avgZ));
}
```

2.3.2 Sortowanie względem dwóch najbliższych wierzchołków

Drugą „wagą", którą postanowiłem zastosować jest suma odległości dwóch najbliższych wierzchołków do kamery.

```
private double getMinimalVerticesDistance(Point point) {
    double min1 = this.vertices.get(0).getDistance(point);
    double min2 = this.vertices.get(0).getDistance(point);
    if (min1 > min2) {
        double tmp = min1;
        min1 = min2;
        min2 = tmp;
    }
    double dist;
    for (int i = 2; i < this.vertices.size(); i++) {
        dist = this.vertices.get(i).getDistance(point);
        if (dist < min1) {
            min2 = min1;
            min1 = dist;
        } else if (dist < min2)
            min2 = dist;
    }
    return min1 + min2;
}
```

2.3.3 Sortowanie względem średniej odległości wszystkich wierzchołków

Ostatnią „wagą", jaką chciałem wypróbować jest średnia odległość wszystkich wierzchołków ściany względem kamery.

```
    private double getAverageVerticesDistance(Point point) {
    double sum = 0;
    for (Point vert: this.vertices) {
        sum += point.getDistance(vert);
    }
    return sum / this.vertices.size();
}
```

2.4 Porównanie wag

Najlepsza okazała się waga określona jako odległość środka ciężkości ściany do punktu, w którym znajduje się kamera. Pozostałe dwie wagi dawały w większości przypadków gorsze wyniki - chociaż w pojedynczych układach obiektów względem kamery zdarzało się z nimi uzyskać lepsze efekty, niż se środkiem ciężkości.