Cloth Simulation Using Verlet Integration in C++ with SFML

Abstract

This paper presents the development of a 2D cloth simulation using Verlet integration, implemented in C++ with the Simple and Fast Multimedia Library (SFML). The simulation models a grid of particles connected by constraints, simulating the physical behavior of cloth under gravity. The system allows for user interaction, enabling the tearing of the cloth by deactivating constraints. This project serves as an educational tool for understanding the principles of cloth simulation and the application of numerical methods in physics-based simulations.

---

1. Introduction

Cloth simulation is a fundamental problem in computer graphics and physics-based animation. Accurate and efficient simulation of cloth behavior is essential for realistic animations in games and movies. Traditional methods involve modeling cloth as a mesh of interconnected particles, with constraints maintaining the distances between them. Verlet integration is a popular numerical method used to update particle positions, offering stability and simplicity.

This project implements a 2D cloth simulation using Verlet integration in C++ with SFML. The simulation models a grid of particles connected by constraints, simulating the physical behavior of cloth under gravity. The system allows for user interaction, enabling the tearing of the cloth by deactivating constraints.

---

2. Mathematical Foundations

2.1 Verlet Integration

Verlet integration is a numerical method used to integrate Newton's equations of motion. It is particularly well-suited for particle-based simulations due to its simplicity and stability. The position update equation is:

$$xt+1=2xt−xt−1+atΔt2$$

where:

- xt is the position at time t,
- $xt−1$ is the position at time t−1,
- at is the acceleration at time t,
- $Δt$ is the time step.

This method does not require explicit velocity calculations and is known for its numerical stability in simulations involving soft bodies.

2.2 Constraints

Constraints are used to maintain the distances between particles, simulating the stretching and compression of the cloth. A common approach is to use distance constraints, which enforce a fixed distance between two particles. The constraint is satisfied by applying a correction to the particles' positions:

$$c=x2−x1length= // c // c×=length−rest\_lengthlengthx1−=c×0.52x2+=c×0.52$$

where:

- x1 and x2 are the positions of the two particles,
- rest_length is the initial distance between the particles.

2.3 Gravity

Gravity is modeled as a constant force applied to each particle:

$$Fgravity=m⋅g$$

where:

- m is the mass of the particle,
- g is the acceleration due to gravity.

In the simulation, we assume unit mass for simplicity, and gravity is applied as a constant force in the downward direction.

---

3. Implementation

3.1 Particle Class

The `Particle` class represents a single particle in the simulation. It stores the position, previous position, acceleration, and a flag indicating whether the particle is pinned (fixed in place). The class provides methods to apply forces, update the particle's position using Verlet integration, and constrain the particle to the bounds of the simulation area.

```
class Particle {
public:
    sf::Vector2f position;
    sf::Vector2f previous_position;
    sf::Vector2f acceleration;
    bool is_pinned;

    Particle(float x, float y, bool pinned = false)
        : position(x, y), previous_position(x, y), acceleration(0, 0), is_pinned(pinned) {}

    void applyForce(const sf::Vector2f& force) {
        if (!is_pinned) {
            acceleration += force;
        }
    }

    void update(float time_step) {
        if (is_pinned) return;
        sf::Vector2f velocity = position - previous_position;
        previous_position = position;
        position += velocity + acceleration * time_step * time_step;
        acceleration = sf::Vector2f(0, 0);
    }

    void constrainToBounds(float width, float height) {
        if (position.x < 0) position.x = 0;
        if (position.x > width) position.x = width;
        if (position.y < 0) position.y = 0;
        if (position.y > height) position.y = height;
    }
};
```

3.2 Constraint Class

The `Constraint` class represents a distance constraint between two particles. It stores pointers to the two particles and the initial distance between them. The class provides a method to satisfy the constraint by applying corrections to the particles' positions.

```
class Constraint {
public:
    Particle* p1;
    Particle* p2;
    float initial_length;
    bool active;

    Constraint(Particle* p1, Particle* p2)
        : p1(p1), p2(p2), initial_length(std::hypot(p2->position.x - p1->position.x, p2->position.y - p1->position.y)), active(true) {}

    void satisfy() {
        if (!active) return;
        sf::Vector2f delta = p2->position - p1->position;
        float current_length = std::hypot(delta.x, delta.y);
        float difference = (current_length - initial_length) / current_length;
        sf::Vector2f correction = delta * 0.5f * difference;

        if (!p1->is_pinned) p1->position += correction;
        if (!p2->is_pinned) p2->position -= correction;
    }

    void deactivate() {
        active = false;
    }
};
```

3.3 InputHandler Class

The `InputHandler` class handles user input, specifically mouse clicks, to allow for the tearing of the cloth. When the user clicks near a constraint, the constraint is deactivated, simulating the tearing of the cloth.

```
class InputHandler {
public:
    static void handle_mouse_click(const sf::Event& event, std::vector<Particle>& particles, std::vector<Constraint>& constraints) {
        if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button == sf::Mouse::Left) {
```

```
            sf::Vector2i mouse_pos = sf::Mouse::getPosition();
            float closest_distance = std::numeric_limits<float>::max();
            Constraint* nearest_constraint = nullptr;

            for (auto& constraint : constraints) {
                if (!constraint.active) continue;
                sf::Vector2f line_start = constraint.p1->position;
                sf::Vector2f line_end = constraint.p2->position;
                sf::Vector2f line_dir = line_end - line_start;
                sf::Vector2f to_mouse = sf::Vector2f(mouse_pos.x, mouse_pos.y) - line_start;
                float line_length = std::hypot(line_dir.x, line_dir.y);
                line_dir /= line_length;
                float projection = to_mouse.x * line_dir.x + to_mouse.y * line_dir.y;
                sf::Vector2f closest_point = line_start + line_dir * projection;
                float distance = std::hypot(closest_point.x - mouse_pos.x, closest_point.y -
mouse_pos.y);

                if (distance < closest_distance) {
                    closest_distance = distance;
                    nearest_constraint = &constraint;
                }
            }

            if (nearest_constraint && closest_distance < CLICK_TOLERANCE) {
                nearest_constraint->deactivate();
            }
        }
    }
};
```

3.4 Main Simulation Loop

The main simulation loop initializes the particles and constraints, applies forces, updates particle positions, satisfies constraints, and handles user input. It also renders the particles and constraints to the screen using SFML.

```
int main() {
    sf::RenderWindow window(sf::VideoMode(WIDTH, HEIGHT), "Cloth Simulation");
    window.setFramerateLimit(60);

    std::vector<Particle> particles;
    std::vector<Constraint> constraints;
```

```
    // Initialize particles and constraints...

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
            }
            InputHandler::handle_mouse_click(event, particles, constraints);
        }

        // Apply gravity and update particles...

        // Satisfy constraints...

        window.clear();
        // Render particles and constraints...
        window.display();
    }

    return 0;
}
```

---

4. Acknowledgments

The development of this cloth simulation was inspired by the tutorial provided by Felipe's Coding on YouTube. The tutorial served as a valuable resource in understanding the principles of cloth simulation and the implementation of Verlet integration. The original video can be found here:

[Cloth Simulation Tutorial by Felipe's Coding]

---

5. Conclusion

This paper presents a simple yet effective implementation of a 2D cloth simulation using Verlet integration in C++ with SFML. The simulation models the physical behavior of cloth under gravity and allows for user interaction, enabling the tearing of the cloth. The project serves as an educational tool for understanding the principles of cloth simulation and the application of numerical methods in physics-based simulations.

Note: The provided code snippets are simplified for clarity. In a complete implementation, additional considerations such as collision detection, performance optimizations, and advanced constraint handling would be necessary for a robust simulation.
