Bird Flock Animation in Rust and CUDA

Objective

Develop two versions of a 3D bird flock animation, one implemented in Rust and the other in CUDA, to simulate natural flocking behaviour using parallel computing techniques.

Background

The goal of this assignment is to implement two versions of parallel programs of a bird flock animation, one in Rust and the other in CUDA. The assignment will assess your understanding of parallel computing concepts, synchronization, Rust's concurrency model and CUDA's parallel computing model.

Bird flocking is a fascinating natural phenomenon where a group of birds flies together in a coordinated manner. This behaviour is not only visually stunning but also serves several purposes, such as protection from predators, efficient navigation, and improved foraging. The flock moves as a cohesive unit, with each bird adjusting its position and velocity based on the movements of its neighbours. It is found that this complex behaviour is mainly governed by three simple rules:

1. separation (avoiding crowding neighbours),
2. alignment (steering towards the average heading of neighbours),
3. and cohesion (moving towards the average position of neighbours).

These simple rules result in complex and dynamic group behaviours, such as flock formation, splitting, merging, and collective evasion from predators.

In this assignment, you will simulate this flocking behaviour by writing two versions of concurrent and parallel programs: one using Rust, the other using CUDA.

By assigning each bird to a separate thread, you can efficiently compute the interactions and movements of a huge number of birds in parallel, resulting in a smooth and realistic animation.

Technical Specifications

The scene comprises a 3D space where multiple birds (agents) interact based on the flocking rules. Each bird has a position, velocity, and acceleration in 3D space.

Algorithm

1. **Initialization:**
   - Initialize the position, velocity, and acceleration of each bird randomly within the 3D space.
2. **Calculate Forces:**
   - For each bird, compute the separation, alignment, and cohesion forces based on nearby birds.
   - Apply additional forces such as gravity and external forces if necessary.

   1) **Separation:**
      - For each bird, calculate the vector pointing away from each nearby bird.
      - Sum these vectors to obtain the separation force.
      - Formula:

$$\text{Separation Force} = -\sum_{i=1}^{N}(\text{Position}_{\text{bird}} - \text{Position}_{\text{neighbour}_i})$$

   2) **Alignment:**
      - For each bird, calculate the average velocity of nearby birds.
      - Steer towards this average velocity.
      - Formula:

$$\text{Alignment Force} = \frac{1}{N}\sum_{i=1}^{N}\text{Velocity}_{\text{neighbour}_i} - \text{Velocity}_{\text{bird}}$$

   3) **Cohesion:**
      - For each bird, calculate the average position of nearby birds.
      - Steer towards this average position.
      - Formula:

$$\text{Cohesion Force} = \frac{1}{N}\sum_{i=1}^{N}\text{Position}_{\text{neighbour}_i} - \text{Position}_{\text{bird}}$$

   4) **Combine Forces:**
      - Combine the separation, alignment, and cohesion forces with appropriate weights.
      - Formula:

$$\text{Total Force} = w_s \cdot \text{Separation Force} + w_a \cdot \text{Alignment Force} + w_c \cdot \text{Cohesion Force}$$

      Where w_s, w_a, and w_c are the weights for separation, alignment, and cohesion forces, respectively.

3. **Update Acceleration:**
    - Sum all forces acting on each bird to determine its acceleration.
    - Formula:

$$\text{Acceleration} = \frac{\text{Total Force}}{\text{Mass}}$$

4. **Update Velocity:**
    - Integrate acceleration to update the bird's velocity.
    - Limit the velocity to a maximum value to prevent unrealistic speeds.
    - Formula:

$$\text{Velocity} = \text{Velocity} + \text{Acceleration} \cdot \Delta t$$

    - Limit velocity:

$$\text{Velocity} = \min(\text{Velocity}, \text{Max Velocity})$$

5. **Update Position:**
    - Integrate velocity to update the bird's position.
    - Ensure the birds remain within the bounds of the 3D space, applying boundary conditions if necessary.
    - Formula:

$$\text{Position} = \text{Position} + \text{Velocity} \cdot \Delta t$$

Parallel Architecture

You are required to implement the animation on two different architectures: Rust and CUDA.

Rust Implementation:

- Divide the flock into segments processed by multiple threads.
- Explore different threading models and compare their performance.

CUDA Implementation:

- Assign each bird to its own CUDA thread.
- Optimize the use of shared memory and synchronization to improve performance.

Visualization

- Render each bird as a simple geometric primitive in 3D space such as a triangle, a sphere, using OpenGL or DirectX.
- Ensure smooth and realistic animation of the flocking behaviour.

Lab Book Entries

1. **GPU Design:**
    - Detail and reflect upon the design of your CUDA implementation, focusing on parallel aspects.
    - Explain the use of threads, mutual exclusion, and synchronization.
2. **CPU Design:**
    - Detail and reflect upon the design of your Rust implementation, focusing on parallel aspects.
    - Explain the use of threads, mutual exclusion, and synchronization.
3. **Performance Metrics:**
    - Compare the performance of your Rust and CUDA implementations, with and without graphics.
    - Describe your performance benchmarks.

Demonstration

- Demonstrate your two solutions in the lab, running on the provided PCs.

Submission

- **Software - CUDA:** Submit your code as a Visual Studio solution, including source code and executables.
- **Software - Rust:** Submit your code as a Visual Studio Code project, including source code and executables.
- **Video:** Provide a short, narrated video showing both implementations.
- **Lab Book:** Submit the full lab book as a PDF.