Equations used for this project

github/piotrkossa/Turing-Pattern-Simulation-In-Unity

Used Source

1 The Project Goal

The goal of this project is to implement the Gray-Scott reaction-diffusion model in real time in Unity. It leverages the parallel processing power of the GPU using Compute Shaders to visualize complex Turing patterns at an efficient refresh rate. This document provides a brief description of the mathematical model and its implementation.

2 Overview

The simulation bases on the interaction between an **Activator** (u) and an **Inhibitor** (v). Everything is driven by a single reaction, where the **Activator** is used up to create more of the **Inhibitor**.

$$u + 2v \rightarrow 3v$$

If that was all, the reaction would stop after a certain amount of time. This is why two processes are constantly active to keep it running:

1. A specified amount of the **Activator** (u) is continuously added to the system, controlled by the **Feed Rate** (f)
2. The **Inhibitor** (v) is slowly removed, controlled by the **Kill Rate** (k)

Obtained patterns come from how the chemicals diffuse on the plane and not the reaction alone. Spreading out at different speeds causes creation of new unique patterns, even after minor change of **Diffusion Rates** (D_u, D_v)

3 Equations of the Gray-Scott Model

$$\partial u/\partial t = (D_u * \nabla^2 u) - (u * v^2) + (f * (1 - u))$$
$$\partial v/\partial t = (D_v * \nabla^2 v) + (u * v^2) - ((k + f) * v)$$

Symbol	Description
u	Concentration of Activator
v	Concentration of Inhibitor
D_u, D_v	Diffusion Rates of Chemicals
$\nabla^2 u, \nabla^2 v$	Laplacian - Spatial Diffusion
f	The Feed Rate, represents a constant supply of substance u
k	The Kill Rate, represents the rate of removal of substance v
$\partial u/\partial t, \partial v/\partial t$	Time Derivatives of Concentrations

4 The Laplacian

The Laplacian operator (∇^2) is calculated using a 3x3 convolution kernel:

$$\begin{bmatrix} 0.05 & 0.20 & 0.05 \\ 0.20 & -1.0 & 0.20 \\ 0.05 & 0.20 & 0.05 \end{bmatrix}$$

For each pixel, the values of its eight neighbors and itself are multiplied by the corresponding weights in the kernel, and the results are summed up to produce the final Laplacian value.

5 Code Implementations

The following code snippets demonstrate how the Gray-Scott model equations are implemented in a custom Unity's Compute Shader.

1. **The Laplacian calculation:**

```
float laplacianU = (
    GetNeighbour(id.xy + int2(0, 1)) * 0.2 +
    GetNeighbour(id.xy + int2(0, -1)) * 0.2 +
    GetNeighbour(id.xy + int2(1, 0)) * 0.2 +
    GetNeighbour(id.xy + int2(-1, 0)) * 0.2 +
    GetNeighbour(id.xy + int2(1, 1)) * 0.05 +
    GetNeighbour(id.xy + int2(1, -1)) * 0.05 +
    GetNeighbour(id.xy + int2(-1, 1)) * 0.05 +
    GetNeighbour(id.xy + int2(-1, -1)) * 0.05 -
    prevU
).x;

float laplacianV = (
    GetNeighbour(id.xy + int2(0, 1)) * 0.2 +
    GetNeighbour(id.xy + int2(0, -1)) * 0.2 +
    GetNeighbour(id.xy + int2(1, 0)) * 0.2 +
    GetNeighbour(id.xy + int2(-1, 0)) * 0.2 +
    GetNeighbour(id.xy + int2(1, 1)) * 0.05 +
    GetNeighbour(id.xy + int2(1, -1)) * 0.05 +
    GetNeighbour(id.xy + int2(-1, 1)) * 0.05 +
    GetNeighbour(id.xy + int2(-1, -1)) * 0.05 -
    prevV
).y;
```

We take each pixel on the texture and calculate the Laplacian for both chemicals using the convolution kernel from The Laplacian section and its neighboring pixels.

2. **The Gray-Scott equations implementation:**

```
float prevU = prevState[id.xy].x;
float prevV = prevState[id.xy].y;

float reaction = prevU * (prevV * prevV);

float deltaU = (diffusionU * laplacianU)
    - reaction
    + (feedRate * (1.0 - prevU));

float deltaV = (diffusionV * laplacianV)
    + reaction
    - ((killRate + feedRate) * prevV);

float newU = clamp(
    prevU + deltaU * deltaTime,
    0.0,
    1.0);

float newV = clamp(
    prevV + deltaV * deltaTime,
    0.0,
    1.0);

newU = clamp(newU, 0.0, 1.0);
newV = clamp(newV, 0.0, 1.0);

Result[id.xy] = float4(newU, newV, 0.0, 1.0);
```

This code calculates the changes in concentrations of both chemicals using the mentioned equations. It updates the concentrations based on diffusion, reaction, feed and kill rates, ensuring the values remain within valid range using clamping function.