МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

УТВЕРЖДАЮ
Зав.кафедрой,
к. ф.-м. н., доцент
_____ А. С. Иванов

ОТЧЕТ О ПРАКТИКЕ

студента 4 курса 451 группы факультета КНиИТ
Григорьева Алексея Александровича

вид практики: преддипломная
кафедра: математической кибернетики и компьютерных наук
курс: 4
семестр: 8
продолжительность: 4 нед., с 30.04.2020 г. по 27.05.2020 г.

Руководитель практики от университета,

доцент _____ М. С. Семенов

Руководитель практики от организации (учреждения, предприятия),

доцент _____ М. С. Семенов

Тема практики:«Безымянная дипломная работа»

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

Паттерн —

Эксперимент (в контексте данной работы) — процесс поэтапной эволюции клеточных автоматов с целью достижения максимального совпадения воспроизводимых ими результатов заданным наперед паттернам.

ВВЕДЕНИЕ

Осознание

Эксперименты по оптимизации проводились для двумерных клеточных автоматов первого порядка, с возможными состояниями 0 и 1. Следующее состояние клеточного автомата определяется 9 соседними автоматами на решетке включая данный или, другими словами, окрестностью Мура порядка 2. Тип автоматов выбран не случайно: количество возможных правил, описывающих их, равно 2^{512}, что гарантирует невозможность перебора. Более того, не существует и общего аналитического решения для получения произвольного изображения, при условии что изначальна двумерная сетка задана случайными значениями. У выбранных в данной работе типов клеточных автоматов также есть преимущества, связанные с возможностью оптимизации алгоритмов. Многие из описанных далее результатов возможно перенести на случай клеточных автоматов N-го порядка, с увеличенным количеством состояний или иными правилами обновления.

Целью данной работы является

1 Окружение для проведения экспериментов

В данном разделе

1.1 Среда разработки

Поставленную задачу решено выполнить с использованием среды разработки `Unity`. Такой выбор основан удобствами, предоставляемыми средой. В частности, на выбор повлияли следующие преимущества:

- быстрая и многофункциональная работа с графикой посредством готовых функций и визуального проектирования;
- профессиональные инструменты для отладки производительности программы;
- наглядность результата программы.

Программный код написан на языке `C#`.

1.2 Реализация клеточного автомата

Опишем правила перехода для двумерного клеточного автомата первого порядка. При подсчете входного сигнала будут учитываться как и состояние данного автомата, так и состояния восьми соседних автоматов в окрестности мура порядка 2, для подсчета берутся значения с последней итерации. Возьмем самый простой случай: автомат может принимать значения 0 и 1. Таким образом, возможны $2^9 = 512$ вариантов входных сигналов. Отсюда и берется упомянутое ранее количество всевозможных правил для данных клеточных автоматов, 2^{512}.

Каждый клеточный автомат будет расположен на двумерной сетке. В границах одной сетки все автоматы имеют единые правила перехода. Ради прозрачности процесса и наглядности результатов, эксперимент проводится полностью в реальном времени: в каждую единицу времени все автоматы на всех сетках переходят в новое состояние, определенное входными сигналами и правилами перехода.

В качестве визуализации клеточных автоматов в `Unity` можно использовать шейдеры. Плюс этого подхода заключается в том, что компьютер не будет тратить ресурсы оперативной памяти для визуализации автоматов, и вся нагрузка на это отдается видеокарте.

1.2.1 Правило перехода

Создадим простейшую реализацию подсчета таблицы перехода для каждого клеточного автомата, и обновления по ней. Для избежания лишних затрат памяти, в основной части программы заранее созданы массивы, для временного хранения состояния автомата на следующем шаге, чтобы полученный результат не влиял на соседние автоматы в данный момент.

Создадим цикл, который будет выполняться на двумерной сетке с клеточными автоматами и размерностью `screenSizeInPixels` по X и Y.

```
var nextCAField = nextVirtualScreens[ind];
var screen2DSize = screenSizeInPixels * screenSizeInPixels;
for (short i = 0; i < screenSizeInPixels; i++)
{
   for (short j = 0; j < screenSizeInPixels; j++)
   {
       int signal = 0;
       for (short k = 0; k < 3; k++)
       {
           for (short m = 0; m < 3; m++)
           {
               signal += CAField[
                   (screen2DSize + screenSizeInPixels * i + j
                   + screenSizeInPixels * (k - 1) + (m - 1))
                   % (screen2DSize)] << (k * 3 + m);
           }
       }
       nextCAField[i * screenSizeInPixels + j] = (byte)allRules[ind][signal];
   }
}

for (short i = 0; i < screen2DSize; i++)
{
   CAField[i] = nextCAField[i];
}
```

1.2.2 Оптимизированное вычисление входных сигналов

1.2.3 Правило перехода на языке шейдеров (HLSL)

Обновление клеточных автоматов возможно полностью перенести в шейдеры. Каждая сетка с клеточным автоматом будет использовать свою текстуру и шейдер.

Для написания правила перехода на языке шейдеров необходимо решить три проблемы:

1. получение информации о состояниях в предыдущий момент времени;
2. соотнесение представления в битах (пикселях) с текстурой в трехмерном пространстве;
3. обновление шейдера только при необходимости.

Возьмем `Custom Render Texture` из среды разработки `Unity`, который создан для отрисовки текстуры с помощью шейдеров. Первая проблема решается расширением языка HLSL, предоставляемой `Custom Render Texture`. В фрагментном шейдере можно обратиться к переменной `localTexcoord`, в которой хранится состояние текстуры на предыдущий момент отрисовки.

Вторая проблема также решается глобальными переменными `_CustomRenderTex` предоставляющая ширину объекта в пространстве и `_CustomRenderTextureHeight`, возвращающая высоту.

Определим настройки `Custom Render Texture` 1. Установка `Wrap Mode` в значение `Repeat` задаст периодические граничные условия для сетки. Другими словами, при обращении к пикселям по координатам, превышающим размерность текстуры, будут взяты пиксели с противоположной стороны сетки. Создадим пустой шейдер, код для которого опишем дальше. Настройка `Update Mode` в значение `On Demand` позволяет обновлять состояние текстуры по вызову функции из основной программы, что позволит получить полный контроль над обновлением автоматов.

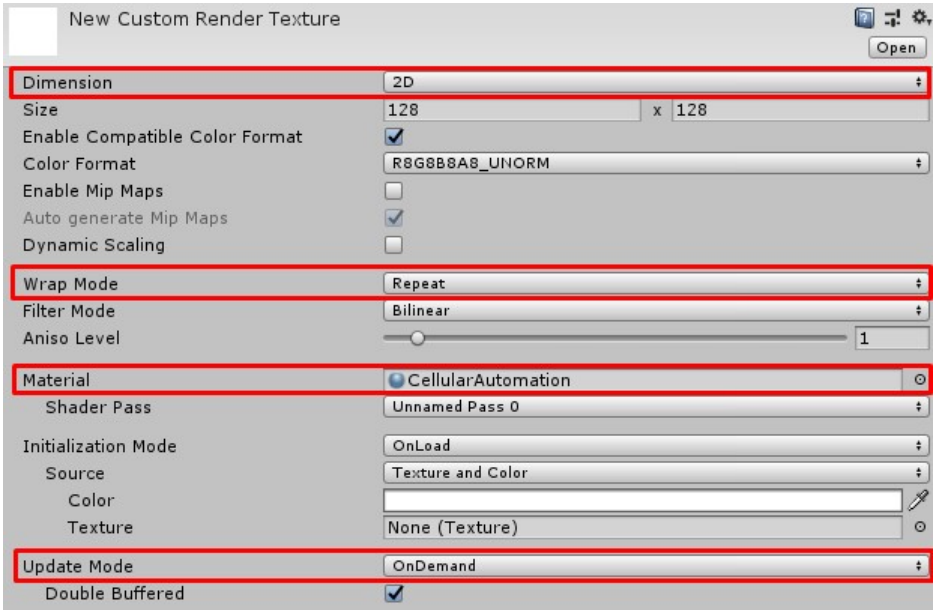

Рисунок 1 – Настройка Custom Render Texture.

Определим uniform-переменную: таблицу перехода размерности 512 (все

возможные состояния рассматриваемого клеточного автомата).

```
float _rule[512];
```

Опишем функцию, возвращающую значение пикселя с заданным отступом в сетке относительно текущего клеточного автомата.

```
float4 get(v2f_customrendertexture IN, int x, int y) : COLOR
{
   return tex2D(_SelfTexture2D, IN.localTexcoord.xy +
            fixed2(x / _CustomRenderTextureWidth, y / _CustomRenderTextureHeight));
}
```

Аналогично построению двоичного числа, последовательно, с левого верхнего угла, соберем биты входных сигналов для клеточного автомата в окрестности Мура порядка 2 и получим следующее состояние из таблицы перехода.

```
float getRule9(v2f_customrendertexture IN) : float
{
   int accumulator = 0;
   for (int i = 2; i >= 0; i--)
   {
       for (int j = 0; j <= 2; j++)
       {
           int roundedAlpha = round(get(IN, i-1, j-1).a);
           accumulator = (accumulator << 1) + roundedAlpha;
       }
   }
   return _rule[accumulator];
}

float4 frag(v2f_customrendertexture IN) : COLOR
{
   return getRule9(IN);
}
```

Осталось объединить шейдер с основной программой. Пусть в массиве `allRules` находятся правила для всех двумерных сеток. Тогда, опуская подробности, инициализацию текстуры можно произвести с помощью следующего кода

```
customRenderTexture.material.SetFloatArray("_rule", allRules[screenInd]);
customRenderTexture.Initialize();
```

Пусть в массиве `custormRenderTextures` содержатся все текстуры (сетки). Тогда, обновить каждый автомат можно, используя следующий фрагмент кода:

```
foreach(var texture in customRenderTextures)
{
   texture.Update();
}
```

2 Генетический алгоритм для поиска клеточных автоматов

В качестве инструмента для получения приближенного результата будет взят генетический алгоритм. **Особи** — клеточные автоматы, **гены** — правила перехода, **приспособленность** особи — степень совпадения некоторой окрестности клеточного автомата некоторому целевому изображению (паттерн), воспроизведение которого и будет являться целью каждого эксперимента.

Для эволюции будут отбираться половина особей с наивысшей приспособленностью, из них случайным образом будут созданы пары, каждая из которых создаст два потомка. Правила переходов для потомков формируются из генов родителей, которые так же сохраняются до следующего этапа эволюции. В работе будут рассмотрены два вида скрещивания.

Для первого вида скрещивания выберем случайный номер бита в генах (правила перехода длиной 512), будем называть его «разделителем». Все биты первого родителя до разделителя станут основой для правила перехода первого дочернего клеточного автомата, все биты с номером $\geq$ разделитель возьмутся из второго родителя. Для второго дочернего клеточного автомата, до разделителя берутся биты из второго родителя, а после — из первого.

Второй вид скрещивания основывается на случайном выборе родителя для каждого бита.

2.1 Подсчет приспособленности

....

При подсчете приспособленности учитывается количество допустимых ошибок в паттерне. Добавляя такой параметр, мы намеренно рискуем получить искаженный результат, но зачастую такая мера необходима для значительного уменьшения времени сходимости.

```
foreach(var texture in customRenderTextures)
{
   texture.Update(); blahblahblah
}
```

В идеальном случае, нам хотелось бы высчитать некоторую нормализованную функцию приспособленности, которая бы для любой сетки и набора паттернов давала бы значение в диапазоне от 0 до 1 (или 100, для наглядности). Подобная функция учитывала бы максимально возможное количество изобра-

жений на паттерне. Таким образом мы могли бы поделить количество вхождений паттерна в изображение на максимально возможное, и измерить процент совпадения. В реальности же это достижимо только с помощью полного перебора всей сетки, что в самом простом случае, при размере изображения 64×64 без оптимизаций требовало бы произвести $2^{64 \times 64}$ подсчетов приспособленности. Поэтому, для упрощения, будем считать значение приспособленности равным 100 если на изображение полностью состоит из паттернов без пересечения. Например, если исходное изображение имеет размерность 64×64, а паттерн — 4×4, то на изображении возможно разместить $64 \times 64 / 4 \times 4 = 256$ паттернов.

Подведем подсчет приспособленности на последних итерациях, и сделаем возможно устанавливать количество итераций перед процессом эволюции. Между двумя подсчетами приспособленности происходит ровно одно обновление клеточных автоматов на сетке.

Если запустим программу с профилировщиком, можно заметить, что самым затратным по времени местом в программе является именно подсчет приспособленности 2. Это не удивительно, ведь для каждого пикселя сетки приходится проверять соответствие окрестности с заданным паттерном, а с увеличением его размера в X раз, в столько же раз увеличивается и количество операций доступа при подсчете приспособленности. Далее попробуем оптимизировать этот процесс.

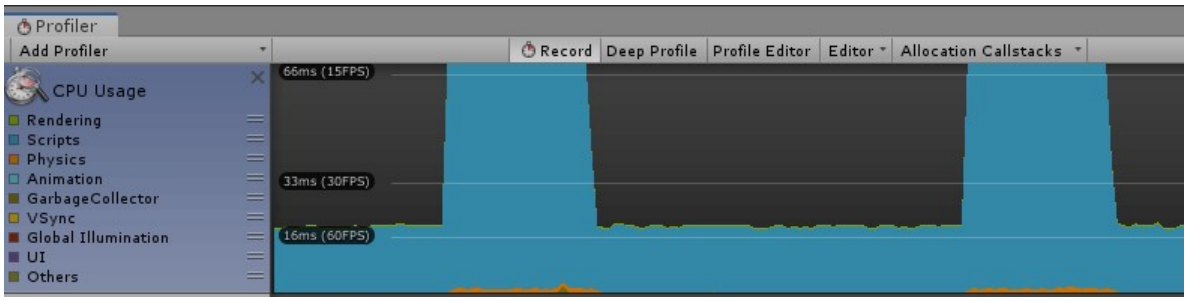

Рисунок 2 – Фрагмент окна профилировщика. Видимые подъемы связаны с процессом подсчета приспособленности

2.2 Подсчет приспособленности на основе данных с GPU

2.3 Оптимизация подсчета приспособленности

2.4 Эволюция

Что-то

2.5 Сбор статистики и визуализация результатов

Что-то

После завершения эксперимента, правила переходов в виде генов записываются в файл, расположенный в пути `{Расположение_проекта}/Assets/Simulati` и имеющий название `G-{Имя_Паттерна}-{IDэксперимента}.txt`. Каждый ген записывается в новой строке, и, соответственно, количество строк в файле определяется числом двумерных сеток в проведенном эксперименте.

Для просмотра клеточных автоматов после эксперимента, создана дополнительная сцена в Unity с названием GeneVisualisation. В основном, она отличается от главной сцены тем, что в ней имеется одна большая сетка размерности 1024×1024, и отключена возможность эволюции автоматов. Для считывания файла с генами создан дополнительный скрипт. В редакторе необходимо выбрать файл с данными 2.5, созданный автоматически после эксперимента. Запустив сцену, можно увидеть, как сетка меняет свое изначальное изображение, используя правила перехода из файла 3. Для переключения между всеми правилами в наборе, необходимо нажать стрелку влево или стрелку вправо на клавиатуре.

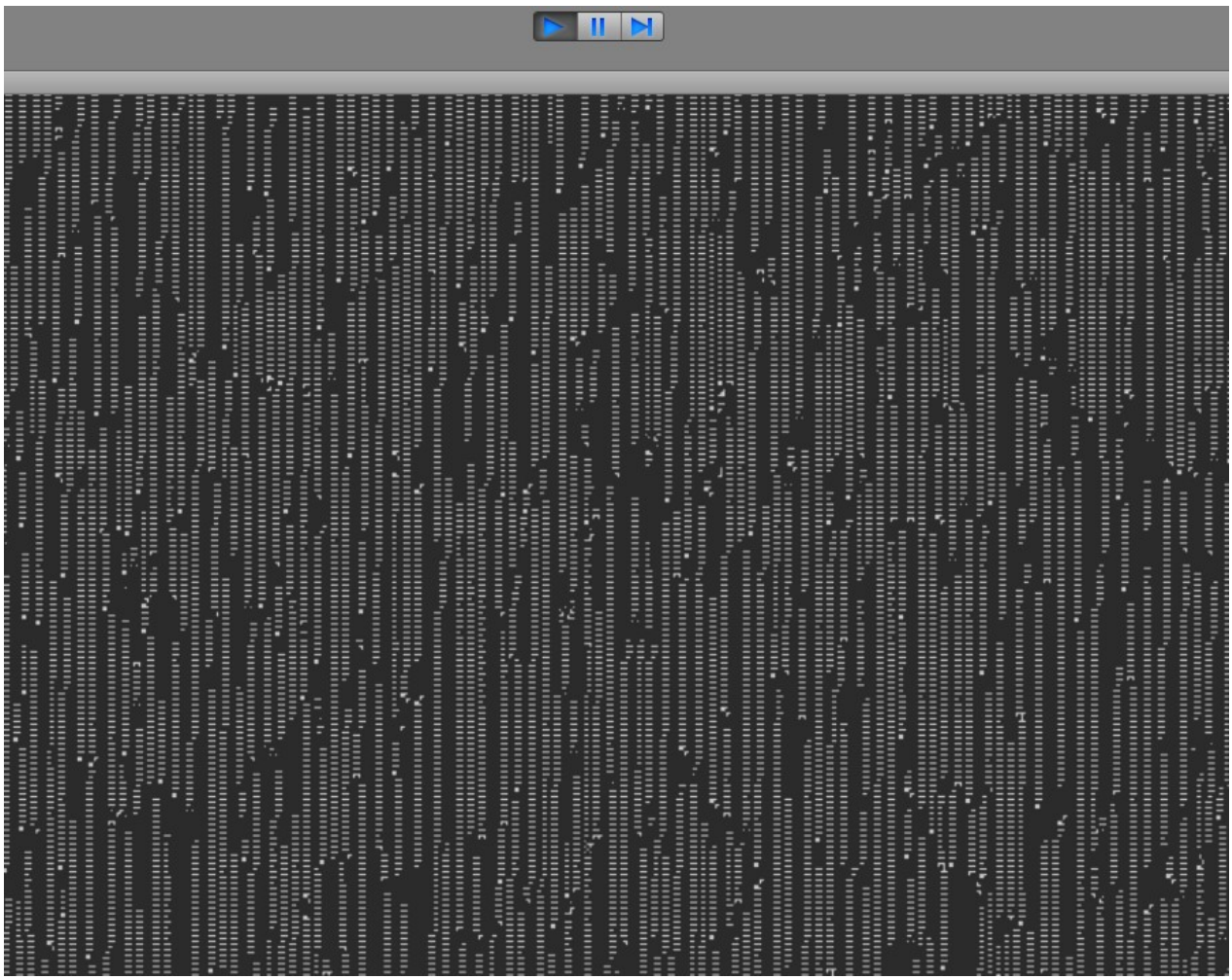

Рисунок 3 – Сцена для визуализации клеточных автоматов после эксперимента. Данный результат получен при попытке воссоздать паттерн «квадрат 3×3 с рамкой в 1 пиксель противоположного цвета.

3 Эксперименты

Данный раздел будет посвящен поиску зависимостей между параметрами эксперимента и скоростью нахождения клеточных автоматов, удовлетворяющих заданным условиям. Возможно ли найти такие параметры модели, которые универсально для любого паттерна будут приводить к быстрому нахождению клеточного автомата, моделирующего его? Если нет, то возможно ли подобрать заранее параметры, наиболее подходящие для заданного паттерна?

3.1 Поиск закономерностей в таблице перехода

Предположим, что для любого паттерна существуют выбранные биты

Более того, можно предположить, что клеточные автоматы в разных экспериментах имеют сходства в правилах перехода.

ЗАКЛЮЧЕНИЕ

Заключение

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А
Листинг программы

?