Визуализация моделирования средствами языка C++ и библиотеки OpenGl на примере клеточного автомата «Игра жизнь»

САМОСТОЯТЕЛЬНАЯ РАБОТА

http://www.kimrt.ru

Онлайн-курс

Суперкомпьютерные технологии в задачах моделирования

Рассмотрим игру «Жизнь» — клеточный автомат, придуманный английским математиком Джоном Конвеем в 1970 году [1]. Место действия этой игры — «вселенная», представляющая собой размеченную на клетки поверхность или плоскость. Она может быть безграничной, ограниченной или замкнутой (в пределе — бесконечная плоскость). Каждая клетка на этой поверхности может находиться в двух состояниях: «живая» (заполненная) или «мёртвая» (пустая). Клетка имеет восемь соседей, окружающих её. Распределение живых клеток в начале игры называется первым поколением. Каждое следующее поколение рассчитывается на основе предыдущего по таким правилам: в пустой (мёртвой) клетке, рядом с которой ровно три живые клетки, зарождается жизнь; если у живой клетки есть две или три живые соседки, то эта клетка продолжает жить; в противном случае, если соседей меньше двух или больше трёх, клетка умирает («от одиночества» или «от перенаселённости»). Следующий код (листинг 1) визуализирует эту игру до

определенного количества поколений, заданного при запуске программы.

Листинг 1. Визуализация клеточного автомата

```
#define _CRT_SECURE_NO_WARNINGS
#include <GL/glut.h>
#include <iostream>
#include "windows.h"
#define _USE_MATH_DEFINES
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <random>
using namespace std;

// размер поля
#define SIZEX 50
#define SIZEY 50
int tmp[SIZEX][SIZEY], world[SIZEX][SIZEY];
int x, y, n = 0;//координаты и количество соседей
int d; // количество поколений
int d_now; // текущее поколение
// кооридинаты для функции отрисовки
float x_p, y_p;

// функция отрисовки в окне
```

```
void display()
{
    if (d_now <= d) {
        // цвет заливки окна
        glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
        // заливка окна
        glClear(GL_COLOR_BUFFER_BIT);
        // размер точки
        glPointSize(19);
        // начало отрисовки точек (клеток)
        glBegin(GL_POINTS);
        for (int i = 0; i < SIZEX; i++)
        {
            for (int j = 0; j < SIZEY; j++)
            {
        // масштабирование координат для отрисовки
                x_p = 2.0/ SIZEX * (i + 1) - 1.02;
                y_p = 2.0/ SIZEY * (j + 1) - 1.02;
    // выбор цвета в зависимости от статуса клетки
                if (world[i][j] == 1) {
                    glColor3d(1.0f, 1.0f, 1.0f);
                }
                else {
                    glColor3d(0.0, 0.0, 0.0);
                }
                // рисование точки (клетки)
```

```
                glVertex2f(x_p, y_p);
            }
        }
        // конец отрисовки точек (клеток)
        glEnd();
        // смена буфера
        glutSwapBuffers();
        // перерисовка окна
        glutPostRedisplay();
    }
}

void Simulation(int value) /*value принимает 3-й
аргумент из glutTimerFunc*/
{
    // расчет следующего поколения
    for (y = 1; y <= SIZEY - 2; y++)
        for (x = 1; x <= SIZEX - 2; x++)
            tmp[x][y] = world[x][y];
    for (y = 1; y <= SIZEY - 2; y++)
    {
        for (x = 1; x <= SIZEX - 2; x++)
        {
            if (tmp[x][y] == 0)
            {
                if (tmp[x - 1][y - 1] == 1) n++;
```

```
            if (tmp[x][y - 1] == 1) n++;
            if (tmp[x + 1][y - 1] == 1) n++;
            if (tmp[x + 1][y] == 1) n++;
            if (tmp[x + 1][y + 1] == 1) n++;
            if (tmp[x][y + 1] == 1) n++;
            if (tmp[x - 1][y + 1] == 1) n++;
            if (tmp[x - 1][y] == 1) n++;
            if (n == 3) world[x][y] = 1;
            n = 0;
        }
        else
        {
            if (tmp[x - 1][y - 1] == 1) n++;
            if (tmp[x][y - 1] == 1) n++;
            if (tmp[x + 1][y - 1] == 1) n++;
            if (tmp[x + 1][y] == 1) n++;
            if (tmp[x + 1][y + 1] == 1) n++;
            if (tmp[x][y + 1] == 1) n++;
            if (tmp[x - 1][y + 1] == 1) n++;
            if (tmp[x - 1][y] == 1) n++;
            if ((n == 2) || (n == 3))
            {
                world[x][y] = 1;
                n = 0;
            }
            else
```

```
                {
                    world[x][y] = 0;
                    n = 0;
                }
            }
        }
    }
    d_now++;
    // повторный вызов таймера
    glutTimerFunc(500, Simulation, 1);

}

int main(int argc, char** argv)
{
    setlocale(LC_ALL, "Russian");
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(1, 10000);
    //начальная генерация
    for (y = 0; y <= SIZEY - 1; y++)
        for (x = 0; x <= SIZEX - 1; x++) {
            int num = dis(gen);
            world[x][y] = num % 2;
        }
    // считывание количества поколений из консоли
```

```
    printf("Количество поколений: ");
    scanf("%i", &d);
    // текущее поколение
    d_now = 1;
    // инициализация glut
    glutInit(&argc, argv);
    // параметры отображения
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
    // размер окна
    glutInitWindowSize(1024, 1024);
    // название окна
    glutCreateWindow("Game");
    // установка фунции отрисовки
    glutDisplayFunc(display);
    // установка таймера
    glutTimerFunc(500, Simulation, 1);
    // запуск обработки событий
    glutMainLoop();
    return 0;
}
```

Рассмотрим код подробнее. В главной функции *main* происходит начальная генерация расположения клеток. После чего в консоли считывается количество поколений, вводимое с клавиатуры. В переменной *d_now* устанавливается значение текущего поколения. Далее происходит взаимодействие с *OpenGL*. Сначала инициализируем взаимодействие с GLUT с помощью команды

glutInit [2, 3]. OpenGL Utility Toolkit (GLUT) — библиотека утилит для приложений под OpenGL, которая в основном отвечает за системный уровень операций ввода-вывода при работе с операционной системой [4]. Эта библиотека является устаревшей, при необходимости можно использовать другие подобные библиотеки, например, freeglut или GLFW. Следующая команда *glutInitDisplayMode* задает параметры отображения: *GLUT_DOUBLE* вывод в окно осуществляется с использованием 2 буферов; *GLUT_RGBA* то же что и RGB, но используется также 4 компонента ALPHA (прозрачность). За размер окна отвечает *glutInitWindowSize*, где первый аргумент – это ширина, а второй – высота в пикселях [5]. Название окна задается с помощью *glutCreateWindow*. Далее указываем на функцию, отвечающую за рисование в окне, с помощью *glutDisplayFunc*. Устанавливаем таймер с помощью *glutTimerFunc*, где первый аргумент задает время в миллисекундах, по истечении которого вызывается функция, второй аргумент указывает на функцию, которую нужно вызвать, третий аргумент – номер таймера, так как таймеров может быть несколько. Также запускаем цикл обработки событий с помощью *glutMainLoop*. Рассмотрим функцию *Simulation* указанную в *glutTimerFunc*. В этой функции происходит расчет следующего поколения по вышеприведенным правилам. В конце функции вызывается *glutTimerFunc* с этой же функцией *Simulation* в качестве одного из параметров для расчета следующего поколения (рекурсия). Функция *display* отвечает за рисование в окне. С помощью *glClearColor(R, G, B, A)* задается цвет и прозрачность фона. Для отчистки буфера до установленных значений

используется *glClear*, а аргумент *GL_COLOR_BUFFER_BIT* указывает на цвет, указанный в *glClearColor* для использования при отчистке буфера. Размер точки задается с помощью *glPointSize*. Для определения границы, внутри которых заданы вершины или группы примитивов, используется *glBegin* и *glEnd*. Агрумент *GL_POINTS* указывает, что будут задаваться координаты точек. С помощью двух циклов перебирается массив с клетками. Далее происходит масштабирование координат, так как в opengl окно имеет систему координат от -1 до 1 по *x* и *y*. Рассмотрим строку `x_p = 2.0 / SIZEX * (i + 1) - 1.02`. Первое число 2.0 длина по x, чтобы узнать соотношение длины в окне OpenGL к 1-й клетке, разделим эту длину на длину массива по *x*. Теперь умножим это отношение на позицию текущей клетки, учитывая, что нумерация элементов массива начинается с нуля. Из координаты вычитается 1.0, чтобы сдвинуть начало координат к -1. Вычитание 0.02 нужно для правильного отображения всех клеток, так как координата клетки располагается в ее центре и расположенные на краю клетки обрезаются краем окна. В зависимости от статуса клетки задается цвет точки с помощью *glColor3d*. После чего *glVertex2f* рисует эту точку (рис. 1). Далее происходит смена буфера на новый *glutSwapBuffers* и перерисовка окна *glutPostRedisplay*.

Также можно ознакомиться с дополнительным материалом: уроки Learn OpenGL [6], уроки Learn OpenGL перевод [7], начало работы с OpenGL [8], инструменты для OpenGL [9].

Рис. 1. Визуализация клеточного автомата «Игра жизнь»

Проект реализуется победителем Конкурса на предоставление грантов преподавателям магистратуры 2020/2021 благотворительной программы «Стипендиальная программа Владимира Потанина» Благотворительного фонда Владимира Потанина.

Источники

1. Игра «Жизнь». URL: https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%B0_%C2%AB%D0%96%D0%B8%D0%B7%D0%BD%D1%8C%C2%BB
2. Уроки по OpenGL с сайта OGLDev. URL: https://triplepointfive.github.io/ogltutor/tutorials/tutorial01.html
3. Описание библиотеки GLUT. URL: https://www.opengl.org.ru/coding/glut/glut2.html

4. GLUT. URL: https://ru.wikipedia.org/wiki/GLUT
5. glutInitWindowPosition, glutInitWindowSize. URL: https://www.opengl.org/resources/libraries/glut/spec3/node11.html
6. Learn OpenGL. URL: https://learnopengl.com/
7. Learn OpenGL перевод. URL: https://habr.com/ru/post/310790/
8. Getting_Started. URL: https://www.khronos.org/opengl/wiki/Getting_Started
9. Context/Window Toolkits. URL: https://www.khronos.org/opengl/wiki/Related_toolkits_and_APIs#Context.2FWindow_Toolkits