Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Компьютерная графика»

Курсовая работа

Тема: Клеточный автомат в 3-х мерном пространстве

Студент: Мукин Юрий
Группа: 80-304
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Написать программу реализующую клеточный автомат в 3-х мерном пространстве. Обеспечить возможность масштабирования и вращения сцены.

2. Описание программы

Программа написана на языке C++. Для визуализации были использованы библиотеки **OpenGL**, **glfw, glew** и **glm.**

- **openGL -** мультиплатформенная библиотека для работы с графикой, предоставляющая обширный инструментарий, который, однако, применен не был.
- **glfw -** фреймворк для работы с openGL в среде языка C++. Позволяет отрисовывать окна и работать с ивентами.
- **glew** - библиотека облегчающая использование методов OpenGL в среде языка C++.
- **glm -** библиотека облегчающая работу с матрицами и векторами. Также типы данных glm полностью совместимы с типами GLSL, что позволяет сильно облегчить подготовку данных для шейдера.

В шейдере была применена модель освещения Фонга.

Алгоритм клеточного автомата - вариация классического алгоритма с полем ограниченным кубом 100x100x100. В качестве параметров алгоритм принимает:

- Промежуток значений количества соседей, при котором клетка может родиться.
- Промежуток значений количества соседей, при котором клетка может выжить.
- Параметр принимающий значение 0/1, которое задает замкнутость поля.
- Массив задающий начальный набор живых клеток, если массив пуст, то живыми окажутся случайные клетки.

На каждой итерации для всех клеток подсчитывается количество живых соседей, на основании этой информации создается массив живых клеток, который передается рендереру.

При запуске программа проверяет наличие файла params.txt,который должен иметь следующую структуру:

- r x1 x2 x3 x4 x5; где x1-5 набор параметров описанный выше. Также этой строки может не быть, в таком случае правила игры будут запрошены у пользователя в консоли.
- p; если в файле есть такая строка то из него будет считан массив положений изначально живых клеток (писать координаты необходимо каждую на отдельной строке через пробел с символом ‘;’ в конце каждой строки).
- в конце последней строки файла должна стоять точка.

Если файл не будет найден, от пользователь будет оповещен об этом, после чего программа запросит правила игры и создает случайный набор изначально живых клеток.

пример структуры файла параметров:

```
r 1 2 2 3 1;
p;
50 50 50;
51 50 50;
49 50 50;
50 51 50;
51 51 50;
49 51 50;
50 49 50;
51 49 50;
```

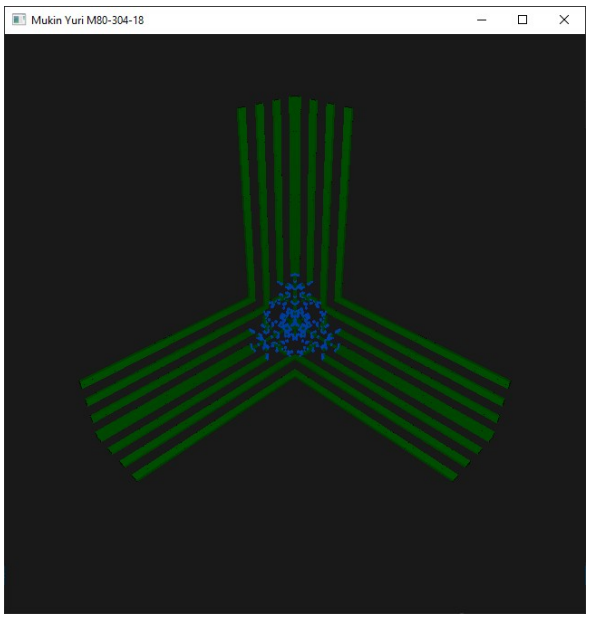
```
49 49 50;
50 50 51;
51 50 51;
49 50 51;
50 51 51;
51 51 51;
49 51 51;
50 49 51;
51 49 51;
49 49 51;
50 50 49;
51 50 49;
49 50 49;
50 51 49;
51 51 49;
49 51 49;
50 49 49;
51 49 49;
49 49 49;.
```

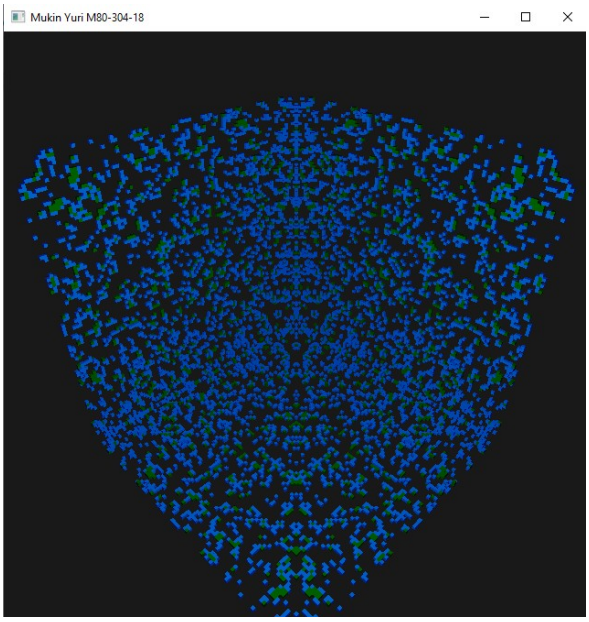
3. Набор тестов

   Пусть в начале будут живы 27 клеток в центре тестового поля. Эти клетки изначально будут иметь конфигурацию куба со стороной 3. Далее применим на этот стартовый набор клеток следующие наборы правил:

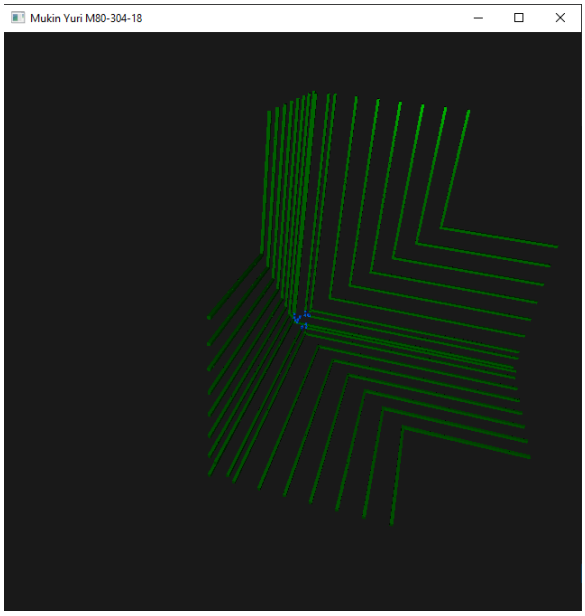
   - тест 1 - 0 1 2 3 1
   - тест 2 - 0 1 2 3 0
   - тест 3 - 1 26 0 26 1
   - тест 4 - 3 4 2 5 0
   - тест 5 - 1 2 2 3 1

4. Результаты выполнения тест

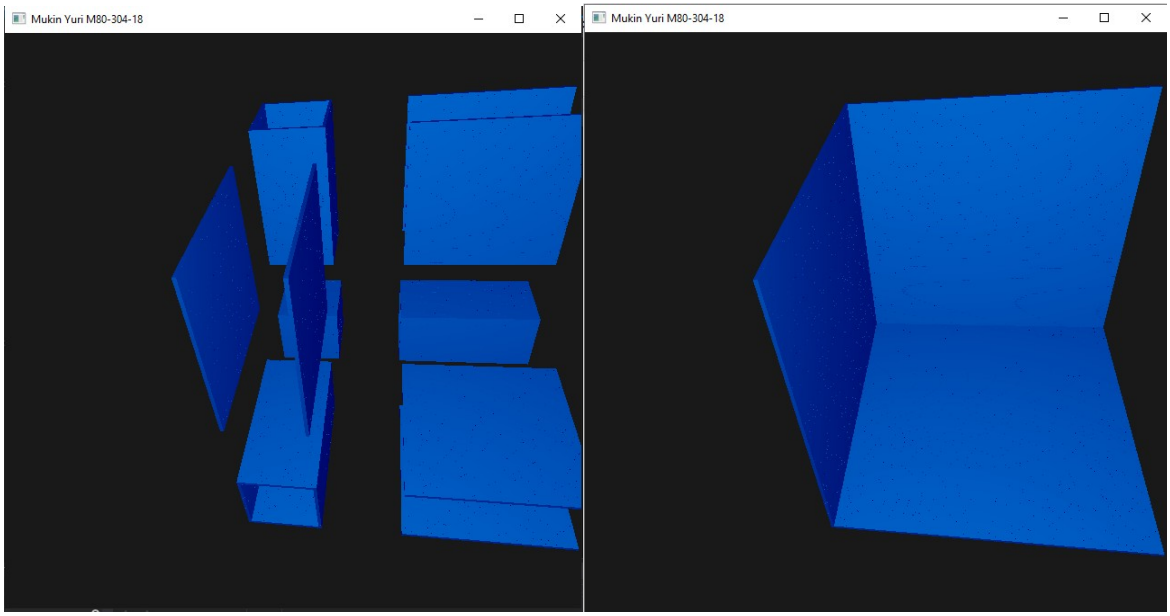




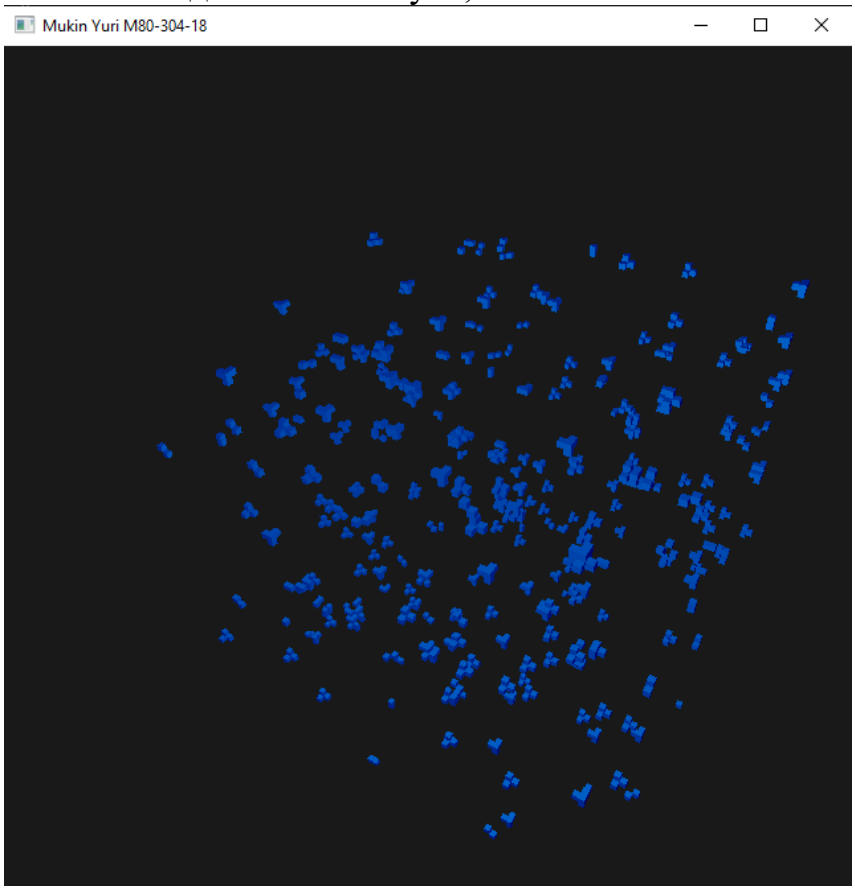
тест 1: В начале система нестабильна и разрастается, но в скором времени она заполняет задние стенки куба осцилляторами с периодом 2, после чего стабилизируется.



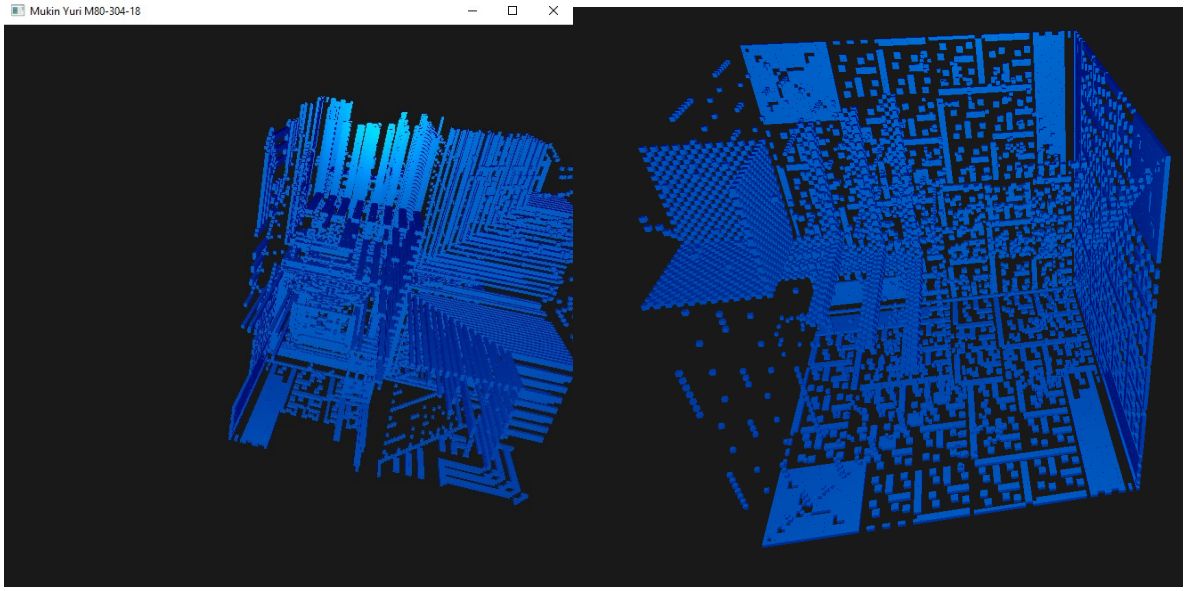
тест 2: При таких парпметрах система является единым осциллятором, который постепенно заполняет все ряды 3-х дальних от наблюдателей стенок куба полосами живых после чего сбрасывается до начального состояния. Стабильность никогда не достигается, период не поддается оценки из-за ограниченного пространства.



тест 3: В этом случае клетки нарастаю огромными пластами, пока не заполнит задние стенки куба, после чего система стабилизируется.



тест 4: Этот набор правил генерирует группу глайдеров, которые довольно быстро уходят за пределы куба.



тест 5: Первые секунды система демонстрирует бурный рост, после чего становится полностью стабильной. Стабильная форма состоит из множества статичных клеток и нескольких десятков осцилляторов с периодом.

5. Листинг программы

Алгоритм клеточного автомата

```
class Game_Life
{
public:
	int NumOfLive;
	int iteration;
	int border;
	Cell Field[101][101][101];
	glm::vec3* AliveArr;

	Game_Life(int Bmin, int Bmax, int Smin, int Smax, Dot inArr[], int ln, bool closed)
	{
		Beath[0] = Bmin; Beath[1] = Bmax;
		Death[0] = Smin; Death[2] = Smax;
		iteration = 1;
		if (closed)
			border = 101;
		else
			border = 100;
		GenStart(inArr, ln);
	}

	void WhoIsAlive()//обновление поля
	{
		std::list<glm::vec3> Live;
		int Neighbors = 0;
		float norm;
		NumOfLive = 0;
		for (int x = 0; x < border; x++)
			for (int y = 0; y < border; y++)
				for (int z = 0; z < border; z++)
				{
					Neighbors = NumOfNeighbors(x, y, z);
					if (Neighbors >= Beath[0] && Neighbors <= Beath[1] &&
!Field[x][y][z].IsLive)
					{
						int xs = x; int ys = y; int zs = z;
```

```
                                                if (x < 100 && y < 100 && z < 100)
                                                {
                                                        Field[x][y][z].nextS = true;
                                                        norm   =   1   /   (sqrt(Neighbors   *
Neighbors * 3));
                                                        Field[xs][ys][zs].collor.x  =  0.001;
Field[xs][ys][zs].collor.y = 0.001; Field[xs][ys][zs].collor.z = Neighbors * norm;
                                                        Live.push_back(Field[x][y]
[z].position);
                                                        NumOfLive++;
                                                }
                                                else
                                                {
                                                        if (x == 100)
                                                                xs = 0;
                                                        if (y == 100)
                                                                ys = 0;
                                                        if (z == 100)
                                                                zs = 0;
                                                        Field[xs][ys][zs].nextS = true;
                                                        norm   =   1   /   (sqrt(Neighbors   *
Neighbors * 3));
                                                        Field[xs][ys][zs].collor.x  =  0.001;
Field[xs][ys][zs].collor.y = 0.001; Field[xs][ys][zs].collor.z = Neighbors * norm;
                                                        Live.push_back(Field[xs][ys]
[zs].position);
                                                        NumOfLive++;
                                                }
                                        }
                                        else  if  (Neighbors  >=  Death[0]  &&  Neighbors  <=
Death[1] && Field[x][y][z].IsLive)
                                        {
                                                int xs = x; int ys = y; int zs = z;
                                                if (x < 100 && y < 100 && z < 100)
                                                {
                                                        Field[x][y][z].nextS = true;
                                                        norm   =   1   /   (sqrt(Neighbors   *
Neighbors * 3));
                                                        Field[xs][ys][zs].collor.x  =  0.001;
Field[xs][ys][zs].collor.y = 0.001; Field[xs][ys][zs].collor.z = Neighbors * norm;
                                                        Live.push_back(Field[x][y]
[z].position);
                                                        NumOfLive++;
                                                }
                                                else
                                                {
                                                        if (x == 100)
                                                                xs = 0;
                                                        if (y == 100)
                                                                ys = 0;
                                                        if (z == 100)
                                                                zs = 0;
                                                        Field[xs][ys][zs].nextS = true;
                                                        norm   =   1   /   (sqrt(Neighbors   *
Neighbors * 3));
                                                        Field[xs][ys][zs].collor.x  =  0.001;
Field[xs][ys][zs].collor.y = 0.001; Field[xs][ys][zs].collor.z = Neighbors * norm;
                                                        Live.push_back(Field[xs][ys]
[zs].position);
                                                        NumOfLive++;
                                                }
                                        }
                                        else  if  (Neighbors  <=  Death[0]  &&  Neighbors  >=
Death[1] && Field[x][y][z].IsLive)
                                        {
                                                norm = 1 / (sqrt(Neighbors * Neighbors * 3));
                                                Field[x][y][z].collor.x  =  Neighbors  *  norm;
Field[x][y][z].collor.y = 0.001; Field[x][y][z].collor.z = 0.001;
                                                Field[x][y][z].nextS = false;
                                        }
                                }
                for (int x = 0; x < 100; x++)
                        for (int y = 0; y < 100; y++)
                                for (int z = 0; z < 100; z++)
```

```
                                        Field[x][y][z].IsLive = Field[x][y][z].nextS;
                delete AliveArr;
                AliveArr = new glm::vec3[NumOfLive];
                for (int l = 0; l < NumOfLive; l++)
                {
                        AliveArr[l] = Live.front();
                        Live.pop_front();
                }
                if (NumOfLive == 0)
                {
                        iteration++;
                        GenStart(NULL,0);
                }
        }

private:
        int Beath[2]; int Death[2];
        int x; int y; int z;

        void GenStart(Dot inArr[], int ln)//генирация стартового набора клеток
        {
                srand(iteration);
                int tmp;
                int count = 0;
                for (int i = 0; i < 100; i++)
                        for (int j = 0; j < 100; j++)
                                for (int k = 0; k < 100; k++)
                                {
                                        if (ln == NULL)
                                        {
                                                tmp = rand() % 10;
                                                if (tmp > 8)
                                                {
                                                        Field[i][j][k].SetPosition(i,  j,  k,
1);
                                                        count++;
                                                }
                                                else
                                                        Field[i][j][k].SetPosition(i,  j,  k,
0);
                                        }
                                        else
                                        {
                                                bool tm = 0;
                                                for (int t = 0; t < ln; t++)
                                                {
                                                        tm   =   ((inArr[t].X   ==   i)   &&
(inArr[t].Y == j) && (inArr[t].Z == k));
                                                        if (tm)
                                                                break;
                                                }
                                                Field[i][j][k].SetPosition(i, j, k, tm);
                                        }
                                }
        }

        int NumOfNeighbors(float x, float y, float z)//подсчет соседей
        {
                int count = 0;
                for (int i = x - 1; i < x + 1; i++)
                        for (int j = y - 1; j < y + 1; j++)
                                for (int k = z - 1; k < z + 1; k++)
                                        if (i >= 0 && i < 100 && j >= 0 && j < 100 && k >= 0
&& k < 100)
                                                if (Field[i][j][k].IsLive)
                                                        count++;
                return count;
        }
};
```

рендерер

```
#define PI 3.14159265

void window_size_callback(GLFWwindow* window, int width, int height);
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset);
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods);
void cursor_position_callback(GLFWwindow* window, double xpos, double ypos);
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods);
double WIDTH, HEIGHT, scale, xposS, yposS, ny, nx;
float x, y, z;
bool isRot;

renderer::renderer(int Bmin, int Bmax, int Smin, int Smax, bool closed, Dot inArr[], int ln)
{
	WIDTH = 640;
	HEIGHT = 640;
	scale = 1;
	xposS = 0;
	yposS = 0;
	nx = 0;
	ny = 0;
	x = 140;
	y = 140;
	z = 140;
	isRot = false;
	Sourse = new Game_Life(Bmin, Bmax, Smin, Smax, inArr, ln, closed);
	renderer::gedCubArr();
	renderer::draw();
}


void renderer::draw()
{
	GLFWwindow* window;// готовим окно и компилируем шейдер
	if (!glfwInit())
		return;
	window = glfwCreateWindow(640, 640, "Mukin Yuri M80-304-18", NULL, NULL);// создаем окно
	if (!window)
	{
		glfwTerminate();
		return;
	}
	glfwMakeContextCurrent(window);
	glLineWidth(3);
	glEnable(GL_DEPTH_TEST);
	Shader                                DifractionShader("../source/shaders/DifractionShader.vs",
"../source/shaders/DifractionShader.frag");
	GLuint VBO, containerVAO;//готовим буферы
	glGenVertexArrays(1, &containerVAO);
	glGenBuffers(1, &VBO);
	glBindBuffer(GL_ARRAY_BUFFER, VBO);
	glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * 216, renderer::CellCube, GL_STATIC_DRAW);
	glBindVertexArray(containerVAO);
	glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
	glEnableVertexAttribArray(0);
	glVertexAttribPointer(1,  3,  GL_FLOAT,  GL_FALSE,  6  *  sizeof(GLfloat),  (GLvoid*)(3  *
sizeof(GLfloat)));
	glEnableVertexAttribArray(1);
	glBindVertexArray(0);
	GLuint lightVAO;
	glGenVertexArrays(1, &lightVAO);
	glBindVertexArray(lightVAO);
	glBindBuffer(GL_ARRAY_BUFFER, VBO);
	glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
	glEnableVertexAttribArray(0);
	glBindVertexArray(0);
	glfwSetScrollCallback(window, scroll_callback); //подписываемся на ивенты
	glfwSetWindowSizeCallback(window, window_size_callback);
	glfwSetMouseButtonCallback(window, mouse_button_callback);
	glfwSetCursorPosCallback(window, cursor_position_callback);
	glfwSetKeyCallback(window, key_callback);
	while (!glfwWindowShouldClose(window)) //бесконечный цикл до закрытия окна
	{
```

```
                glfwPollEvents();
                (*(renderer::Sourse)).WhoIsAlive();
                glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
                glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
                float skf[] = { scale,0,0,0,0,scale,0,0,0,0,scale,0,0,0,0,1 };//считаем трансформации
                glm::mat4 sk = glm::make_mat4(skf);
                float rtxf[] = { 1,0,0,0,0,cos(nx),-sin(nx),0,0,sin(nx),cos(nx),0,0,0,0,1 };
                glm::mat4 rtx = glm::make_mat4(rtxf);
                float rtyf[] = { cos(-ny),0,sin(-ny),0,0,1,0,0,-sin(-ny),0,cos(-ny),0,0,0,0,1 };
                glm::mat4 rty = glm::make_mat4(rtyf);
                DifractionShader.Use(); //используем шейдер
                GLint objectColorLoc = glGetUniformLocation(DifractionShader.Program, "objectColor");
                GLint lightColorLoc = glGetUniformLocation(DifractionShader.Program, "lightColor");
                GLint lightPosLoc = glGetUniformLocation(DifractionShader.Program, "lightPos");
                GLint viewPosLoc = glGetUniformLocation(DifractionShader.Program, "viewPos");
                glUniform3f(lightColorLoc, 0.0f, 0.5f, 0.5f);
                glUniform3f(lightPosLoc, 120.0f,  120.0f,  120.0f);
                glUniform3f(viewPosLoc, x, y, z);
                glm::mat4 view;
                view = glm::lookAt(glm::vec3(x, y, z), glm::vec3(50, 50, 50), glm::vec3(0, 1, 0));
                glm::mat4 projection = glm::perspective(45.0f, (GLfloat)WIDTH / (GLfloat)HEIGHT, 0.1f,
1100.0f);
                GLint modelLoc = glGetUniformLocation(DifractionShader.Program, "model");
                GLint viewLoc = glGetUniformLocation(DifractionShader.Program, "view");
                GLint projLoc = glGetUniformLocation(DifractionShader.Program, "projection");
                glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
                glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
                glBindVertexArray(containerVAO);
                glm::mat4 model = sk  * rtx * rty;
                for (GLuint i = 0; i < (*(renderer::Sourse)).NumOfLive; i++)
                {
                        glm::vec3          collor          =          (*(renderer::Sourse)).Field[(int)
(*(renderer::Sourse)).AliveArr[i].x][(int)(*(renderer::Sourse)).AliveArr[i].y][(int)
(*(renderer::Sourse)).AliveArr[i].z].collor;//считываем цвет клетки
                        glUniform3f(objectColorLoc, collor.x,collor.y,collor.z);
                        glm::mat4            model_loc            =            glm::translate(model,
(*(renderer::Sourse)).AliveArr[i]);//задаем позиция через смещение
                        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model_loc));
                        glDrawArrays(GL_TRIANGLES, 0, 36);
                }
                glBindVertexArray(0);
                glfwSwapBuffers(window);
        }
        glfwTerminate();
}


void renderer::gedCubArr()
{
        GLfloat tmp[] = {
         -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
         0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
         0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
         0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
        -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
        -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,

        -0.5f, -0.5f,  0.5f,  0.0f,  0.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  0.0f,  0.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  0.0f,  0.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  0.0f,  0.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,  0.0f,  0.0f, 1.0f,
        -0.5f, -0.5f,  0.5f,  0.0f,  0.0f, 1.0f,

        -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
        -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
        -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
        -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
        -0.5f, -0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
        -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,

         0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
         0.5f,  0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
         0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
```

```
         0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
         0.5f, -0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
         0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,

        -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
         0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
         0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
         0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
        -0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
        -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,

        -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
         0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
         0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
         0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
        -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
        -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f
        };
        renderer::CellCube = new GLfloat[216];
        for(int i =0; i<216; i++)
                renderer::CellCube[i] = tmp[i];
}

void window_size_callback(GLFWwindow* window, int width, int height)// реагируем на изменение размеров
окна
{
        WIDTH = width;
        HEIGHT = height;
        glViewport(0, 0, width, height);
}

void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
        if (yoffset < 0)// изменяем масштаб 1 шаг колесика мыши =  +/-10 процентов
                scale *= 0.9;
        else
                scale *= 1.1;
}

void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
        if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS)// регаируем на зажатие лкм
        {
                isRot = true;
                glfwSetInputMode(window, 0, 0);
                glfwGetCursorPos(window, &xposS, &yposS);
        }
        if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_RELEASE)
        {
                isRot = false;
                xposS = 0;
                yposS = 0;
        }
}

void cursor_position_callback(GLFWwindow* window, double xpos, double ypos)
{
        if (isRot)// если лкм зажата считаем угол поворота относительно x и y
        {
                if ((xpos / 2000 <= 1) && (ypos / 2000 <= 1))
                {
                        ny += (acos(xpos / 2000) - acos(xposS / 2000)) * 180.0 / PI;
                        nx += (asin(ypos / 2000) - asin(yposS / 2000)) * 180.0 / PI;
                        xposS = xpos; yposS = ypos;
                }
                else
                {
                        isRot = false;
                        xposS = 0;
                        yposS = 0;
                }
        }
}
```

```
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
	if (key == GLFW_KEY_W && action == GLFW_PRESS)
	{
		x -= x * 0.05; y -= y * 0.05; z -= z * 0.05;
	}
	if (key == GLFW_KEY_S && action == GLFW_PRESS)
	{
		x += x * 0.05; y += y * 0.05; z += z * 0.05;
	}
	if (key == GLFW_KEY_D && action == GLFW_PRESS)
	{
		x += z * 0.05; z -= x * 0.05;
	}
	if (key == GLFW_KEY_A && action == GLFW_PRESS)
	{
		x -= z * 0.05; z += x * 0.05;
	}
}
```

Список литературы

1. Статья на сайте **wikipedia**. URL: https://en.wikipedia.org/wiki/Cellular_automaton
2. Документация **glfw**. URL: https://www.glfw.org/documentation.html.
3. Руководство **openGL**. URL: https://www.opengl.org.ru/docs/pg/index.html