

КАФЕДРА Системы автоматизированного проектирования (РК-6)

по дисциплине: «Компьютерная графика»

Вариант лабораторной работы 2

Оценка _____

Москва, 2022 г.

ОГЛАВЛЕНИЕ

ЦЕЛЬ РАБОТЫ	3
Задание	3
Вводная часть	4
Разбор кода.....	6
Результаты работы программы.....	11
ВЫВОДЫ	12
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	12

ЦЕЛЬ РАБОТЫ

Цель лабораторной работы – реализовать визуализацию движущегося по графу объекта на основе PhysX Tutorials, используя встроенные визуальные примитивы; для поиска кратчайшего пути из начальной точки графа в конечную, реализовать алгоритм Дейкстры.

Задание

В среде визуализации трёхмерной графики на основе PhysX Tutorials, используя встроенные визуальные примитивы, реализовать визуализацию движущегося по графу объекта.

Для этого задачу декомпозировать на следующие:

- а) Генерация графа: вершины графа – точки регулярной прямоугольной сетки в плоскости XY. Для каждой точки задаются случайные смещения offsetX, offsetY, offsetZ, значения которых меньше половины шага сетки. В программе граф хранится в удобной для разработчика форме;
 - б) Реализация алгоритма Дейкстры;
 - в) Движение объекта по графу.
- Объект при старте программы появляется в одной из вершин графа.
 - Выбирается конечная точка «путешествия» для объекта на графе.
 - Ищется кратчайший путь к этой точке.
 - Объект продолжает движение до достижения цели.
 - Выбирается новая точка назначения.

Объект путешествует по графу бесконечно.

По графу может «путешествовать» несколько объектов. Объекты могут проходить сквозь друг друга, не представляя препятствий для движения.

Объект представляется шаром.

Граф рисуется в виде обычных линий белого цвета.

Найденный кратчайший путь рисуется с помощью стрелок заранее выбранного цвета, отличного от белого.

В настроечном файле должны быть доступны следующие параметры:

graphPointsCountX, graphPointsCountY – количество точек графа по осям;

objectsCount – количество движущихся объектов;

offsetX, offsetY, offsetZ - предельные значения случайных смещений точек графа относительно регулярной сетки;

objectVelocity – скорость движения объекта (в произвольных абстрактных единицах измерения).

Вводная часть

Для реализации программы использовался алгоритм Дейкстры – алгоритм на графах, который находит кратчайшие пути от одной из вершин графа до всех остальных.

Работает он следующим образом:

1. Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a .
2. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки.
3. Работа алгоритма завершается, когда все вершины посещены.

Инициализация:

1. Метка самой вершины a полагается равной 0, метки остальных вершин — бесконечности, что говорит о том, что расстояния от a до других вершин пока неизвестны.
2. Все вершины графа помечаются как непосещённые.

Шаги алгоритма:

1. Если все вершины посещены, алгоритм завершается.
2. В противном случае, из ещё не посещённых вершин выбирается вершина u , имеющая минимальную метку.
3. Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, в которые ведут рёбра из u , назовём соседями этой вершины. Для каждого соседа вершины u , кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом.
4. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещённую и повторим шаг алгоритма.
5. Алгоритм заканчивает работу, когда все вершины посещены.
6. Если в какой-то момент все непосещённые вершины помечены бесконечностью, то это значит, что до этих вершин нельзя добраться (то есть граф несвязный). Тогда алгоритм может быть завершён досрочно.

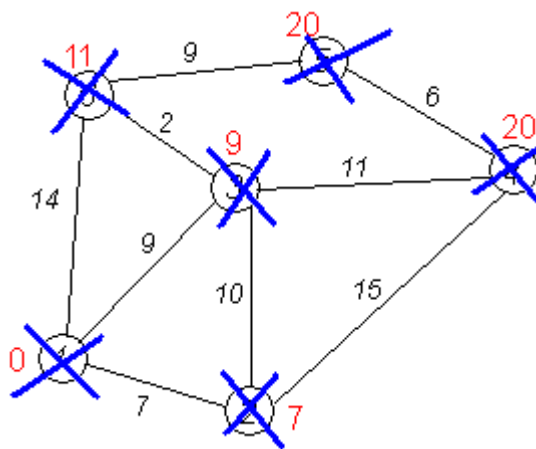


Рисунок 1. Конечно состояние алгоритма, после прохода всех вершин при перемещении из вершины 1 в любую другую

Результат работы алгоритма виден на рисунке 1: кратчайший путь от вершины 1 до 2-й составляет 7, до 3-й — 9, до 4-й — 20, до 5-й — 20, до 6-й — 11.

Для алгоритма Дейкстры мы используем одномерный массив (вектор). Изначально в этом массиве значения всех вершин, кроме начальной — бесконечность, значение начальной — 0.

На каждом шаге, мы выбираем не помеченную вершину с наименьшим значением и ищем из неё пути в другие вершины. Далее прибавляем вес ребра к весу первой вершины и сверяем получившееся значение со значением 2-ой вершины в массиве весов. Если сумма меньше, чем значение в массиве, то присваиваем в массив полученное значение. Помечаем вершину, как пройденную. Повторяем алгоритм, пока не дойдём до целевой вершины.

Разбор кода

Чтение входных данных производим из XML файла:

```
42 void ReadXMLData() {
43     TiXmlDocument doc;
44     string str;
45     ofstream filedot;
46
47     int check;
48
49     if (!doc.LoadFile("./settings.xml"))
50         exit(100);
51
52     TiXmlElement* element = doc.FirstChildElement("root");
53     const char* textNode = element->FirstChild("graphPointsCountX")->FirstChild()->Value();
54
55     str = textNode;
56     graphPointsCountX = stoi(str, nullptr, 10);
57
58     str = element->FirstChild("graphPointsCountY")->FirstChild()->Value();
59     graphPointsCountY = stoi(str, nullptr, 10);
60
61     str = element->FirstChild("objectsCount")->FirstChild()->Value();
62     objectsCount = stoi(str, nullptr, 10);
63
64     str = element->FirstChild("offsetX")->FirstChild()->Value();
65     offsetX = stof(str, nullptr);
66
67     str = element->FirstChild("offsetY")->FirstChild()->Value();
68     offsetY = stof(str, nullptr);
69
70     str = element->FirstChild("offsetZ")->FirstChild()->Value();
71     offsetZ = stof(str, nullptr);
72
73     str = element->FirstChild("objectVelocity")->FirstChild()->Value();
74     objectVelocity = stof(str, nullptr);
75 }
```

Инициализация объекта Шар происходит с помощью класса *Ball*:

```

32 class Ball {
33 public:
34     float start = getCurrentTime();
35     Ball(NxActor* _body) {
36         body = _body;
37         color = 0;
38     }
39     void setPath(std::vector<int> _path) {
40         bpath = _path;
41     }
42     void setColor(int _color) {
43         color = _color;
44     }
45     void move() {
46         if (!bpath.empty()) {
47             NxVec3 p1 = points[bpath[bpath.size() - 1]];
48             NxVec3 p2 = points[bpath[bpath.size() - 2]];
49             NxVec3 lastPos = body->getGlobalPosition();
50             NxVec3 newPos;
51             NxVec3 colorForArrow(0, 1, 1);
52
53             float end = getCurrentTime();
54             float deltaTime = end - start;
55             float length = sqrt((p2.x - p1.x)*(p2.x - p1.x) + (p2.y - p1.y)*(p2.y - p1.y) + (p2.z - p1.z)*(p2.z - p1.z));
56
57             float alpha = acos((p2.x - p1.x) / length);
58             float betta = acos((p2.y - p1.y) / length);
59             float gamma = acos((p2.z - p1.z) / length);
60             bool xb = p2.x > p1.x;
61             bool yb = p2.y > p1.y;
62             bool zb = p2.z > p1.z;
63             if (xb && (lastPos.x > p2.x || lastPos.x < p1.x) || !xb && (lastPos.x > p1.x || lastPos.x < p2.x)
64                 &&
65                 yb && (lastPos.y > p2.y || lastPos.y < p1.y) || !yb && (lastPos.y > p1.y || lastPos.y < p2.y)
66                 &&
67                 zb && (lastPos.z > p2.z || lastPos.z < p1.z) || !zb && (lastPos.z > p1.z || lastPos.z < p2.z))
68             {
69                 newPos = p2;
70                 int index1 = bpath[bpath.size() - 1];
71                 int index2 = bpath[bpath.size() - 2];
72                 bpath.pop_back();
73
74                 for (int j = 0; j < (int)ribs.size(); ++j) {
75                     if ((ribs[j][0] == index1 && (ribs[j][1] == index2
76                         || (ribs[j][1] == index1 && (ribs[j][0] == index2) {
77                         (ribs[j][2] -= color);
78                     }
79                 }
80
81                 if (bpath.size() < 2) {
82                     int endVertex = rand() % points.size();
83                     while (bpath[0] == endVertex) endVertex = rand() % points.size();
84                     setPath(path(bpath[0], endVertex));
85                     painted(bpath, color);
86                 }
87             }
88             else {
89                 newPos = NxVec3(lastPos.x + cos(alpha)*objectVelocity*deltaTime,
90                     lastPos.y + cos(betta)*objectVelocity*deltaTime,
91                     lastPos.z + cos(gamma)*objectVelocity*deltaTime);
92             }
93             for (int i = 0; i < (bpath.size() - 2); i++) {
94                 DrawArrow(points[bpath[i + 1]], points[bpath[i]], getColor(color));
95             }
96             body->setGlobalPosition(newPos);
97             DrawArrow(body->getGlobalPosition(), p2, getColor(color));
98             DrawLine(p1, body->getGlobalPosition(), getColor(273), 3.0f);
99             start = getCurrentTime();
100         }
101     }
102 private:
103     NxActor* body;
104     int color;
105     std::vector<int> bpath;
106 };

```

Отслеживание нажатий на клавиши для смены ракурса камеры:

```

160 void ProcessCameraKeys()
161 {
162     NxReal deltaTime;
163
164     if (bPause)
165         deltaTime = 0.0005;
166     else
167         deltaTime = gDeltaTime;
168
169     // Process camera keys
170     for (int i = 0; i < MAX_KEYS; i++) {
171         if (!gKeys[i]) { continue; }
172
173         // Camera controls
174         switch (i)
175         {
176             case 'w': { gCameraPos += gCameraForward * gCameraSpeed*deltaTime; break; }
177             case 's': { gCameraPos -= gCameraForward * gCameraSpeed*deltaTime; break; }
178             case 'a': { gCameraPos -= gCameraRight * gCameraSpeed*deltaTime; break; }
179             case 'd': { gCameraPos += gCameraRight * gCameraSpeed*deltaTime; break; }
180             case 'z': { gCameraPos += NxVec3(0, 1, 0)*gCameraSpeed*deltaTime; break; }
181             case 'q': { gCameraPos += NxVec3(0, 1, 0)*gCameraSpeed*deltaTime; break; }
182         }
183     }
184 }

```

Функция отрисовки графа:

```

621 void DrawGraph(std::vector<NxVec3> points, std::vector< std::vector<int> > ribs) {
622     for (int i = 0; i < (int)ribs.size(); ++i) {
623         int p1 = (ribs[i])[0];
624         int p2 = (ribs[i])[1];
625         int color = (ribs[i])[2];
626         if (color == 273)
627             DrawLine(points[p1], points[p2], getColor(color), 3.0f);
628     }
629 }

```

Функция генерации графа:


```

641 void GenerateGraph(std::vector<NxVec3>& points, std::vector< std::vector<int> >& ribs, int graphPointsCountX, int graphPointsCountY) {
642     int countVert = 0;
643
644     if (graphPointsCountX < 2)
645         return;
646
647     for (int i = 0; i < graphPointsCountX; ++i) {
648         for (int j = 0; j < graphPointsCountY; ++j) {
649
650             float randX = dc(offsetX);
651             float randZ = dc(offsetZ);
652             float randY = dc(offsetY);
653
654             points.push_back(NxVec3(1.0f * j + randX, offsetZ * 2 + randZ, 1.0f * i + randY));
655         }
656     }
657
658     int color = 273;
659
660     for (int i = 0; i < graphPointsCountX - 1; ++i) {
661         for (int j = 0; j < graphPointsCountY - 1; ++j) {
662
663             std::vector<int> temp;
664
665             temp.push_back(graphPointsCountY*i + j);
666             temp.push_back(graphPointsCountY*i + j + 1);
667             temp.push_back(color);
668             int temp1, temp2;
669             if (temp1 = (rand() % 12)) ribs.push_back(temp);
670
671             temp.clear();
672             temp.push_back(graphPointsCountY*i + j);
673             temp.push_back(graphPointsCountY*i + j + graphPointsCountY);
674             temp.push_back(color);
675             if ((temp2 = (rand() % 4)) || !temp1) ribs.push_back(temp);
676
677             temp.clear();
678             temp.push_back(graphPointsCountY*i + j);
679             temp.push_back(graphPointsCountY*i + j + graphPointsCountY + 1);
680             temp.push_back(color);
681             if ((rand() % 2) || !temp1 && !temp2) ribs.push_back(temp);
682
683         }
684     }
685     for (int j = 0; j < graphPointsCountY - 1; ++j) {
686         std::vector<int> temp;
687
688         temp.clear();
689         temp.push_back(graphPointsCountY * (graphPointsCountX - 1) + j);
690         temp.push_back(graphPointsCountY * (graphPointsCountX - 1) + j + 1);
691         temp.push_back(color);
692         ribs.push_back(temp);
693     }
694 }

```

Функция инициализации матрицы, по которой происходит инициализация графа:

```

696 void GetGraphMatrix(std::vector<NxVec3>& points, std::vector< std::vector<int> >& ribs) {
697
698     // Matrix initialization
699     std::vector<double> row;
700
701     for (int i = 0; i < (int)points.size(); ++i) {
702         row.clear();
703
704         for (int j = 0; j < (int)points.size(); ++j) {
705             row.push_back(0);
706         }
707
708         matrix.push_back(row);
709     }
710
711     for (int i = 0; i < (int)ribs.size(); ++i) {
712         int p1 = (ribs[i])[0];
713         int p2 = (ribs[i])[1];
714         double x1, x2, y1, y2, z1, z2;
715
716         x1 = points[p1].x;
717         x2 = points[p2].x;
718         y1 = points[p1].y;
719         y2 = points[p2].y;
720         z1 = points[p1].z;
721         z2 = points[p2].z;
722
723         (matrix[p1])[p2] = sqrt((x2 - x1)*(x2 - x1) + (y2 - y1)*(y2 - y1) + (z2 - z1)*(z2 - z1));
724         (matrix[p2])[p1] = sqrt((x2 - x1)*(x2 - x1) + (y2 - y1)*(y2 - y1) + (z2 - z1)*(z2 - z1));
725     }
726 }

```

Алгоритм Дейкстры реализован в функции *DijkstraAlgorithm*:

```

742 std::vector<int> DijkstraAlgorithm(int start) {
743     int N = matrix.size();
744
745     std::vector<float> labels;
746     std::vector<bool> isVisits;
747     std::vector<int> path;
748
749     for (int i = 0; i < N; ++i) {
750         labels.push_back(1e+6);
751         isVisits.push_back(false);
752         path.push_back(start);
753     }
754
755     labels[start] = 0;
756
757     std::priority_queue < float, std::vector<float>, std::greater<float> > vertexs;
758
759     int indexVertex = start;
760
761     do {
762         for (int i = 0; i < N; ++i) {
763             float h = (matrix[indexVertex])[i];
764
765             if (h == 0) continue;
766             if (h + labels[indexVertex] < labels[i]) {
767                 labels[i] = h + labels[indexVertex];
768                 path[i] = indexVertex;
769             }
770         }
771         isVisits[indexVertex] = true;
772         indexVertex = minElement(labels, isVisits);
773     } while (indexVertex != labels.size() - 1);
774
775     return path;
776 }

```

Результаты работы программы

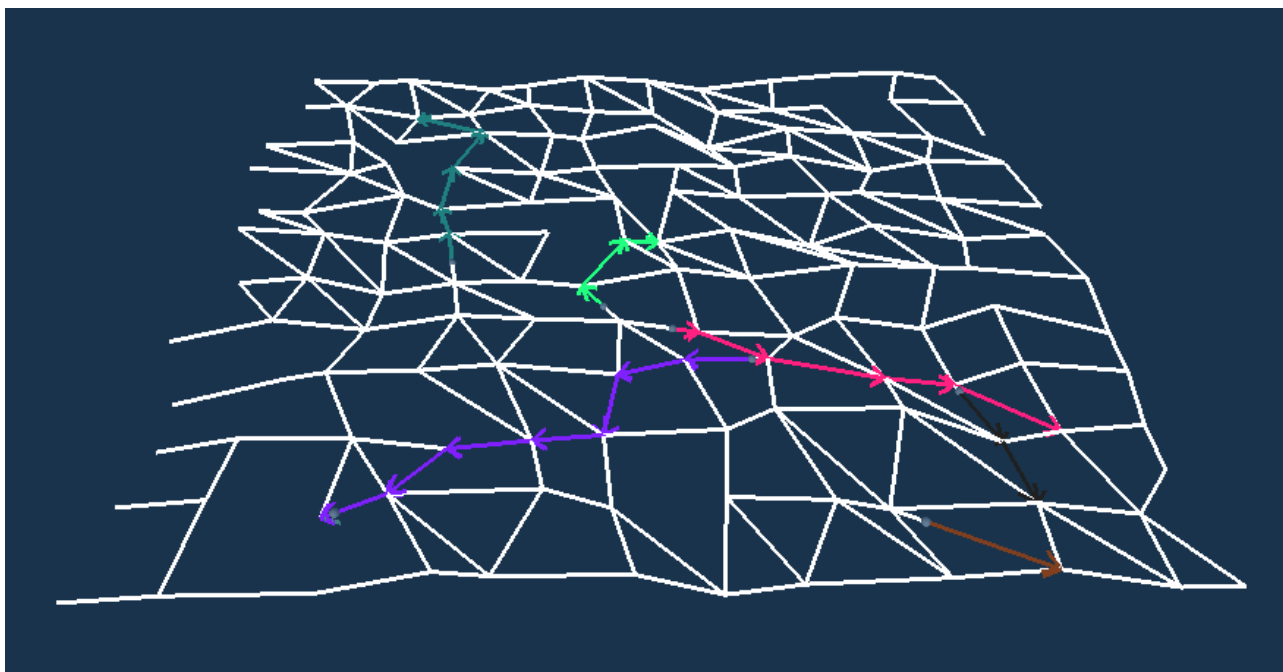


Рисунок 2. Полученная визуализация при $\text{graphPointsCountX} = 12$,
 $\text{graphPointsCountY} = 12$, $\text{objectsCount} = 7$

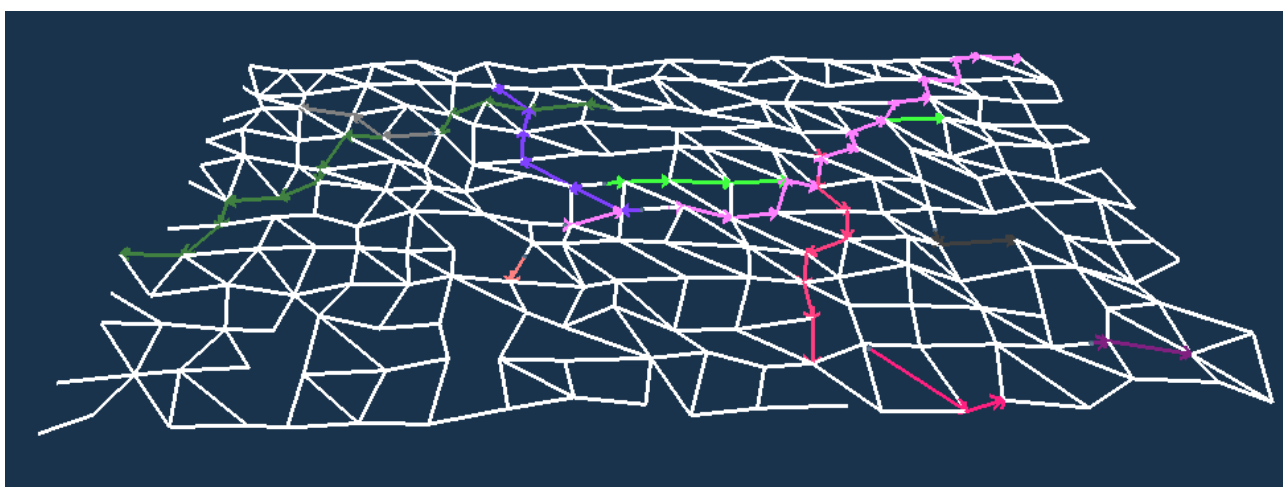


Рисунок 3. Полученная визуализация при $\text{graphPointsCountX} = 12$,
 $\text{graphPointsCountY} = 20$, $\text{objectsCount} = 10$

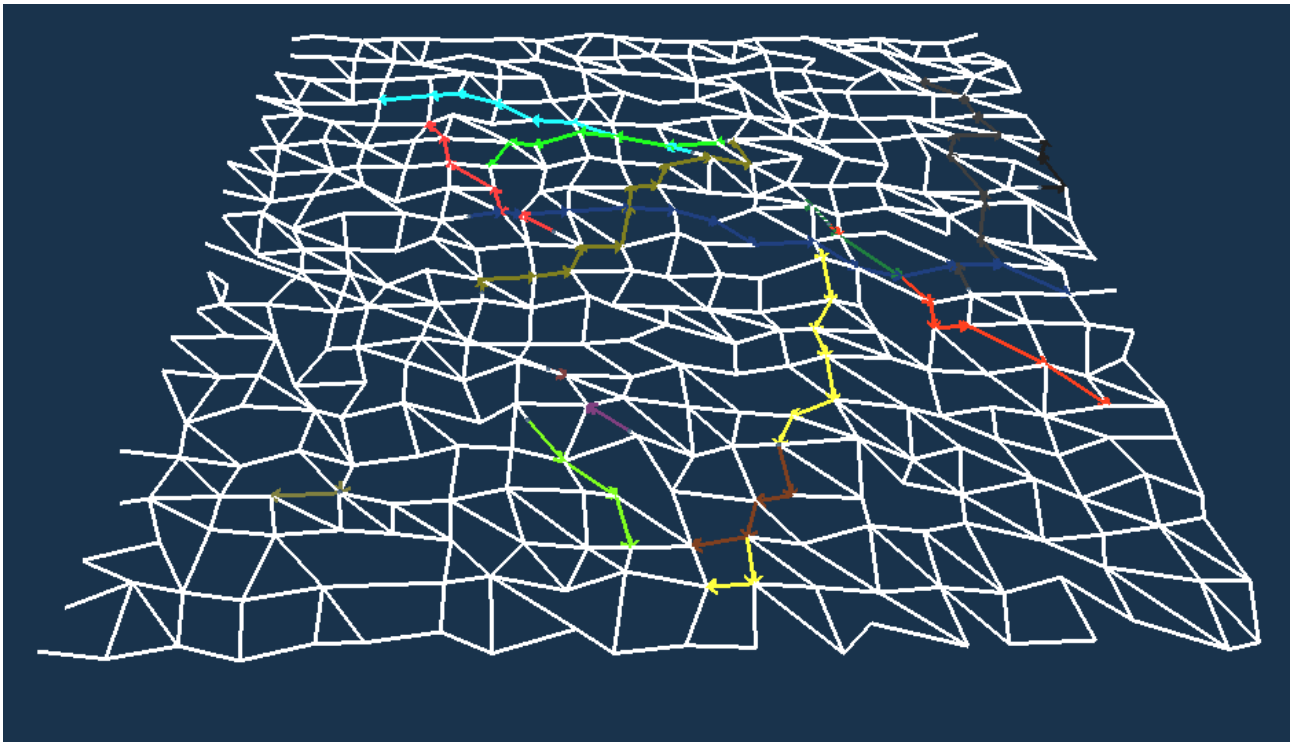


Рисунок 4. Полученная визуализация при $\text{graphPointsCountX} = 20$,
 $\text{graphPointsCountY} = 20$, $\text{objectsCount} = 15$

ВЫВОДЫ

В данной лабораторной работе была реализована визуализация движущегося по графу объекта на основе PhysX Tutorials с использованием встроенных визуальных примитивов, также был реализован алгоритм Дейкстры для поиска кратчайшего пути из начальной точки графа в конечную.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Алгоритм Дейкстры // Википедия URL: https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D1%8B (дата обращения: 06.06.2022).
2. Алгоритм Дейкстры. Поиск оптимальных маршрутов на графе // Хабр URL: <https://habr.com/ru/post/111361/> (дата обращения: 06.06.2022).
3. PhysX SDK. Rigid Body chapter, lessons 101-116. Nvidia Corporation, 2008.