электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.925.2.021

Программный эмулятор перемещения по лабиринту

Сёмина В. А., студент
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение ЭВМ и информационные технологии»

Научный руководитель: Ломовской И.В., старший преподаватель
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
ilomovskoy@mail.ru

Введение

В качестве задания на курсовой проект по дисциплине “Компьютерная графика” была выбрана задача эмуляции перемещения по трехмерному лабиринту. Для решения подобного рода задач часто используется подход, впервые использованный в компьютерной игре *Wolfenstein 3D* [2]. В основе такого подхода лежит алгоритм «бросания лучей» (англ. *ray casting*), который позволят трехмерную задачу свести к двумерной. За счет этого алгоритм демонстрирует высокую производительность. Однако недостатком данного алгоритма является то, что невозможны стены произвольной конфигурации.

В рамках курсового проекта задача эмуляции перемещения по лабиринту решалась в трехмерном пространстве. При этом освещение рассчитывалось с учетом произвольной конфигурации источников и построения теней. Решение задачи в трехмерном пространстве позволило предусмотреть наличие окон в стенах и стены произвольной конфигурации.

Общий алгоритм работы программы и недостатки его реализации

Для представления лабиринта была выбрана полигональная модель [3].

В программе лабиринт представляется в виде набора стен, пола и потолка, а также источников освещения.

Каждая стена описывается набором граней. Пол и потолок – это стена, состоящая из одной грани. Текстуры для стен, пола и потолка хранятся в отдельном массиве.

Все источники освещения являются точечными и обладают следующими характеристиками: интенсивность освещения, радиус освещения, положение в трехмерном пространстве.

В качестве алгоритма удаления невидимых линий и поверхностей был выбран алгоритм Z – буфера [4], так как он прост в реализации, справляется со сценами любой сложности, его оценка вычислительной трудоемкости не более чем линейна.

Ниже приведен общий алгоритм работы программы:

1) Предварительное заполнение текстур стен, пола и потолка лабиринта.
2) Предварительное освещение текстур стен, пола и потолка лабиринта.
3) Преобразование массива граней лабиринта (смещение и поворот).
4) Трехмерное отсечение граней, попавших в область видимости героя лабиринта.
5) Триангуляция граней, попавших в область видимости героя.
6) Перспективное проецирование треугольников.
7) Растеризация треугольников.
8) Вывод на экран полученного изображения.
9) Обработка нажатий на клавиши для учета перемещения по лабиринту.
10) Переход на шаг 3.

Первые запуски программы показали ее низкую производительность. В качестве меры «производительность» использовалась кадровая частота (англ. *Frame Per Second*).

Для обнаружения «узких» мест в алгоритме было проведено инструментирование программы с помощью утилиты *gprof* [5]. Инструментирование выполнялось для конфигурации лабиринта, представленной на рисунке 1.

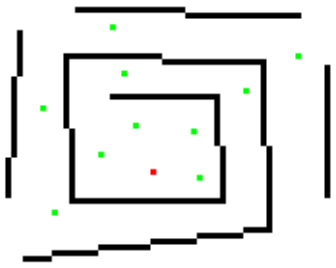

Рис. 1. Конфигурация лабиринта, выбранная для исследования производительности программы

До проведения оптимизации среднее значение *FPS* для конфигурации лабиринта, показанной на рисунке 1, составляло 0,6.

Оптимизация общего алгоритма работы программы

На первом шаге оптимизации было решено изменить способ вывода сформированного изображения лабиринта на экран. Вместо привычного вывода изображения на экран (в цикле по размеру картинной плоскости) был реализован способ, в основе которого лежит использование класса *QImage* для представления матрицы цветов пикселей. Согласно документации *Qt* [6], *QImage* разработан и оптимизирован для прямого доступа к пикселям и манипуляций над ними. Содержимое *QImage* отображается на экран с помощью класса *QLabel* с предварительной конвертацией к типу данных *QPixmap*. *QPixmap* является закадровым представлением изображения, которое может быть использовано в качестве устройства рисования.

Изменение способа вывода изображения увеличило величину *FPS* примерно в два раза. Для конфигурации лабиринта, представленной на рисунке 1, среднее значение количества кадров в секунду стало равным 1,1.

Основной идеей второго шага оптимизации является реализация предварительного текстурирования пола, потолка и стен лабиринта. При этом для повышения производительности текстурирование стен совмещается с расчетом их освещенности. Из-за большого расхода памяти последнее было решено не делать для пола и потолка.

Реализация идеи состоит в хранении для пола и потолка набора заранее рассчитанных изображений, которые выводятся на экран. Количество изображений зависит от приращения угла поворота, которое задается в настройках программы, и вычисляется по формуле:

$$n = \frac{360^\circ}{2 * \Delta_{arc}} \qquad (1),$$

где n – количество изображений, необходимое для представления пола и потолка, Δ_{arc} – приращение угла поворота.

В знаменателе формулы (1) записано $2 * \Delta_{arc}$. Деление на двойку делается с целью сокращения количества изображений. Так можно поступить в силу симметрии пола и потолка (рисунок текстуры на потолке и полу перед героем и сзади него идентичен).

Грани, из которых состоят стены лабиринта, были разделены на четыре типа:

1) Грань, лежащая в плоскости *XOZ*.

2) Грань, лежащая в плоскости *XOY*.

3) Грань, лежащая в плоскости *ZOY*.

4) Грань общего положения.

Размер текстуры должен полностью соответствовать размеру грани, следовательно:

1) Для случая, когда грань лежит в плоскости *XOZ*, требуется текстура размера [*delta_z*][*delta_x*], где *delta_z* и *delta_x* – разница между максимальным и минимальным значением координат *z* и *x* соответственно.

2) Для всех остальных случаев (грань лежит в плоскости *XOY* или в плоскости *ZOY* или грань общего положения) требуется текстура размера [*delta_y*][$\sqrt{delta\_x^2 + delta\_z^2}$].

Для реализации текстурирования были получены формулы для расчета координат трехмерной точки по индексам пикселя в матрице цветов текстуры и получения индексов пикселя в матрице цветов текстуры, соответствующего координатам трехмерной точки. Данные формулы получаются из обычной математической пропорции.

Несомненно, число источников в сцене будет влиять на производительность программы, и, следовательно, на количество кадров формируемого изображения в секунду. Таким образом, чем больше количество источников освещения в сцене, тем ниже будет скорость работы программного продукта.

Кроме того, учитывая, что количество изображений, необходимое для пола и потолка, зависит от изменения угла поворота, то объем требуемой памяти будет также зависеть от изменения угла поворота.

На рисунке 2 показан график зависимости требуемой памяти от количества изображений. По горизонтальной оси графика отложено количество изображений, по вертикальной – объем требуемой памяти в Мб.

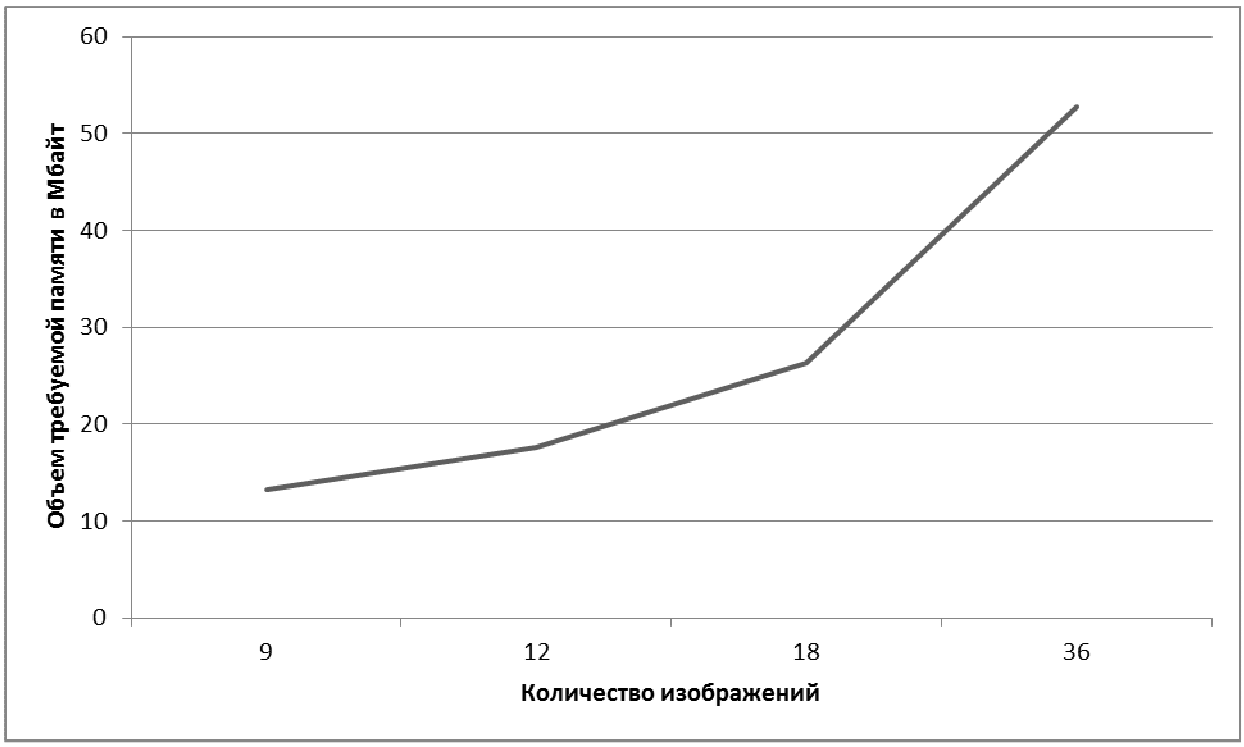

Рис. 2. График зависимости требуемой памяти от количества изображений

В результате проведения второго шага оптимизации среднее значение *FPS* выросло в 1,5 раза.

На третьем шаге оптимизации для пола и потолка было реализовано предварительное освещение, как и для стен, несмотря на большие затраты памяти.

Несомненно, на предварительное освещение потолка и пола затрачивается значительное количество времени, но, благодаря тому, что освещение пола и потолка теперь происходит один раз, скорость работы программного эмулятора перестала зависеть от количества источников в сцене.

В результате последнего шага оптимизации среднее значение *FPS* для конфигурации лабиринта, представленной на рисунке 1, выросло примерно в 12 раз.

Для конфигурации лабиринта, изображенной на рисунке 1, были построены графики зависимости среднего значения *FPS* от количества источников освещения в сцене для проекта до оптимизации и после каждого шага оптимизации. Полученные графики представлены на рисунке 3.

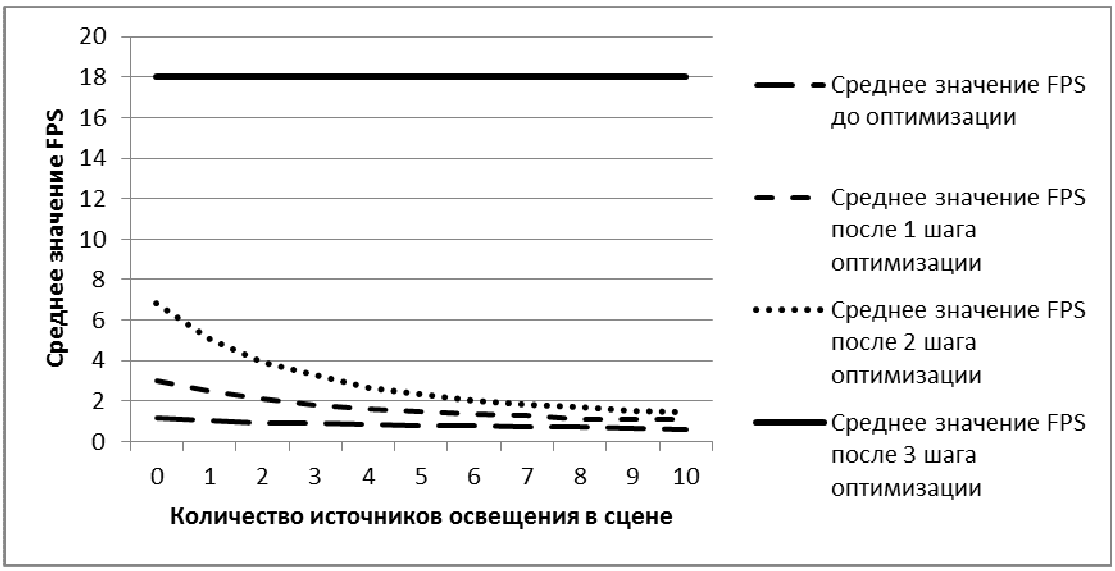

Рис. 3. Графики зависимости среднего значения *FPS* от количества источников освещения в сцене

Как уже было сказано, предварительное текстурирование стен, пола и потолка требует большого объема оперативной памяти. Расходы памяти, необходимые для каждого шага оптимизации, представлены на рисунке 4.

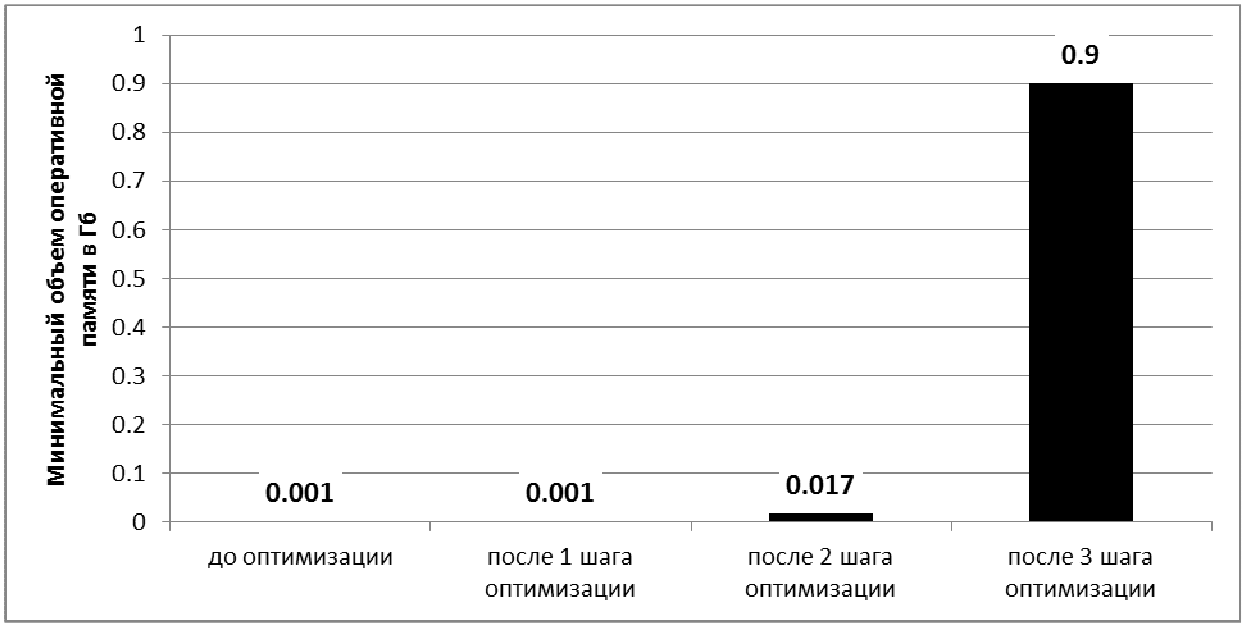

Рис. 4. График зависимости объема требуемой оперативной памяти в Гб от шага оптимизации

Заключение

Задача перемещения по лабиринту была решена в трехмерном пространстве. Это позволило учесть освещенность, тени, окна в стенах лабиринта, наличие стен произвольной конфигурации. Была проведена оптимизация разработанного решения, что позволило добиться высокой производительности и приемлемого значения *FPS* (кадровая частота порядка 20). Выигрыш в скорости получен за счет значительного увеличения объема используемой памяти.

Список литературы

1. Роджерс Д. Алгоритмические основы машинной графики (English Translation: Matematical Elements for Computer Graphics). М.: Мир, 1989. 512 с.
2. Wolfenstein 3D. Режим доступа: https://ru.wikipedia.org/wiki/Wolfenstein_3D (дата обращения 25.03.2015).
3. Полигональное моделирование. Режим доступа: https://ru.wikipedia.org/wiki/Полигональное_моделирование (дата обращения 25.03.2015).
4. Алгоритм, использующий Z – буфер. Режим доступа: http://compgraph.tpu.ru/zbuffer.htm (дата обращения 25.03.2015).
5. GNU gprof. Режим доступа: https://sourceware.org/binutils/docs/gprof/ (дата обращения 25.03.2015).
6. QImage Class Reference. Available at: http://doc-snapshots.qt.io/4.8/qimage.html accessed 25.03.2015.
7. Гасов В.М., Михеев В.А., Черненький В. М. Виды трехмерной графики. Аналитическая и сплайновая трехмерные графики. М.: Изд-во МГТУ им. Н. Э. Баумана, 2012. 32 с.