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

УДК 004.94

Алгоритмы и решения, использованные при разработке программы моделирования фигур вращения

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

Научный руководитель: Волкова Л.Л., ассистент
кафедра «Программное обеспечение ЭВМ и информационные технологии»,
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
irudakov@bmstu.ru

С появлением компьютеров одной из главных задач, вставшей перед программистами, была задача 3D моделирования. Сегодня компьютерная графика применяется в самых разных сферах жизни человека, от игровой индустрии до моделирования в медицине и машиностроении. Помимо этого в наши дни все большую популярность приобретает 3D печать. Существует множество программ, позволяющих спроектировать фигуру для дальнейшей ее отправки на печать, такие как AutoCAD, 3DS Max и т.д., но большинство этих программ являются платными. Следовательно, на данный момент является актуальной задача разработки бесплатного ПО для моделирования трехмерных фигур.

В рамках курсового проекта по машинной графике было разработано ПО, решающее выше поставленную задачу на примере фигур вращения и включающее:

1) программу создания модели фигуры вращения с любой образующей и степенью триангуляции;

2) программу отображения текстурированных фигур на шахматной доске, освещаемой разнородными источниками освещения.

В статье описываются идеи и решения, использованные при разработке программы, приводятся результаты тестирования на производительность при различных входных параметрах.

Для разработки программы необходимо было решить следующие задачи:

1. выбрать методы создания модели фигуры вращения любой формы и степени триангуляции;

2. проанализировать существующие алгоритмы расчета текстурных координат и при необходимости разработать новый;

3. выбрать и модифицировать алгоритмы и методы отображения объектов на сцене, в частности:

3.1. алгоритм удаления невидимых граней;

3.2. алгоритм растеризации треугольников;

3.3. механизм изменения интенсивности цвета, заданного в цветовой схеме RGB;

4. разработать общую последовательность видовых преобразований на основе выбранных алгоритмов;

5. оптимизировать разработанную программу.

Рассмотрим каждую из этих задач подробнее.

1. Метод создания фигур вращения

При разработке программы создания модели фигур вращения необходимо было выбрать математические методы, с помощью которых можно будет составить кривую вращения, которая не будет зависеть от какой - либо из ортогональных осей X или Y.

Для решения задачи независимости от ортогональных осей необходимо выбрать кривые, которые можно задавать в параметрической форме. Так же необходимо было предоставить возможность, как интерполяции, так и аппроксимации точек, заданных на плоскости для генерации образующей фигуры вращения. В качестве таких кривых было решено использовать кривые Безье [4] и интерполяцию Катмулл-Рома [2]. Кривые Безье являются одними из самых известных способов аппроксимации точек в кривую. Главным преимуществом данного способа аппроксимации является ее простота в реализации и хорошая плавность получаемой кривой. В свою очередь, интерполяция Катмулл-Рома является параметрической формой кубической интерполяции. Описанные выше средства позволяют создавать кривую вращения любой формы.

2. Алгоритм расчета текстурных координат

Одной из самых сложных проблем, вставших при разработке программы, являлась задача расчета текстурных координат. Стоит заметить, что до сих пор не существует универсального алгоритмического решения задачи наложения текстуры на тело произвольной формы. В настоящее время основным способом наложения текстур является uv развертка.

Главная идея uv развертки [5] заключается в предварительном расчете в программе создания модели фигуры текстурных координат для вершин каждого треугольника, из которых состоит модель, и последующей интерполяции для получения текстурной координаты в конкретном пикселе. Отсюда вытекает главная проблема uv развертки: необходимость расчета uv координат таким образом, чтобы произошло равномерное наложение текстуры. Рассмотрим несколько алгоритмов uv развертки.

1) Кубическая развертка. Основная идея заключается в том, что мы имеем 6 разных текстур (по количеству сторон куба).

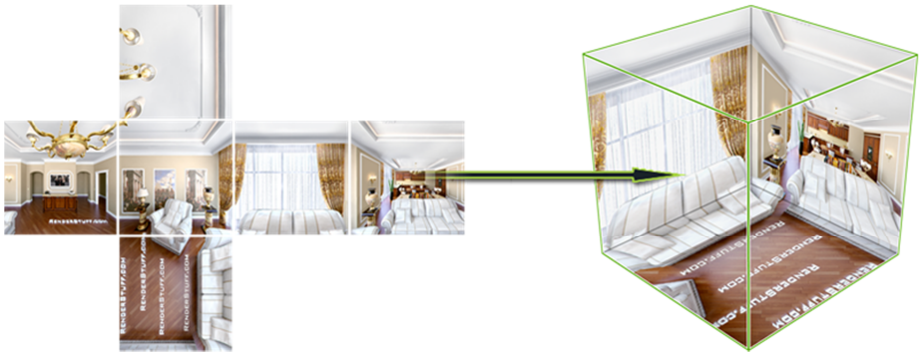

Рис. 1. Принцип работы кубической развертки

На рис. 1 можно увидеть основной недостаток данного вида текстур – необходимо иметь текстуру с каждого из 6 направлений обзора.

2) Цилиндрическая развертка. Основная идея заключается в преобразовании параллелей в горизонтали (см. рис. 2).

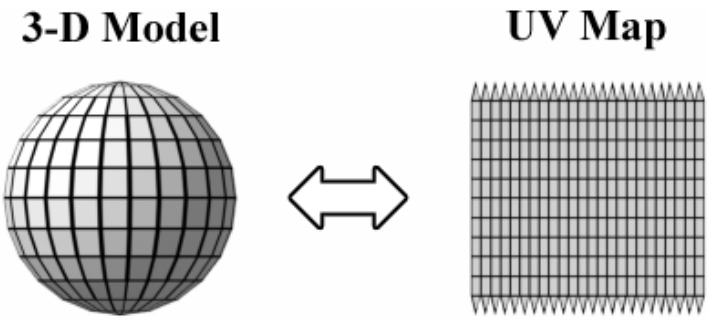

Рис. 2. Принцип работы цилиндрической развертки

Т.к. программа разрабатывалась для моделирования фигур вращения, то в качестве алгоритма uv развертки целесообразно было бы взять алгоритм цилиндрической развертки с распрямлением образующей фигуры вращения и последующим дублированием полученного отрезка с определенных шагом по X, как это происходит при обычной цилиндрической развертке (см. рис. 3).

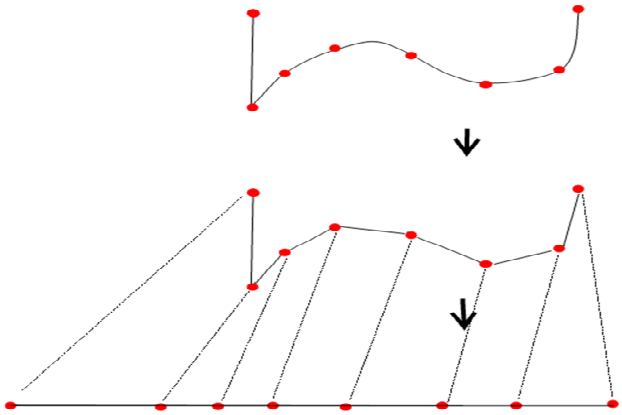

Рис. 3. Принцип работы цилиндрической развертки с выпрямлением образующей фигуры вращения

3. Алгоритмы и методы отображения объектов на сцене.

Рассмотрим основные алгоритмы отображения объектов на сцене.

3.1. Алгоритм удаления невидимых граней

Из всего множества алгоритмов удаления невидимых частей фигур было решено в качестве основного выбрать алгоритм, использующий z-буфер, – он достаточно прост в реализации, имеет возможность распараллеливания и легко переносится на аппаратную составляющую. Суть этого алгоритма заключается в том, что глубина каждого нового отрисовываемого пикселя сравнивается с текущим значением глубины в z-буфере. Если координата z нового пикселя ближе к наблюдателю, то обновляем значение глубины в данном пикселе z-буфера и заносим цвет данного пикселя в буфер кадра.

Основной принцип работы параллельной реализации алгоритма, использующего z-буфер, заключается в разбиении экрана на n параллельных y-групп (см. рис. 4) и создании n потоков, каждый из которых обрабатывает только пиксели своей группы. Таким образом, обеспечивается решение проблемы монопольного доступа к данным z-буфера и буфера кадра при параллельной реализации алгоритма.

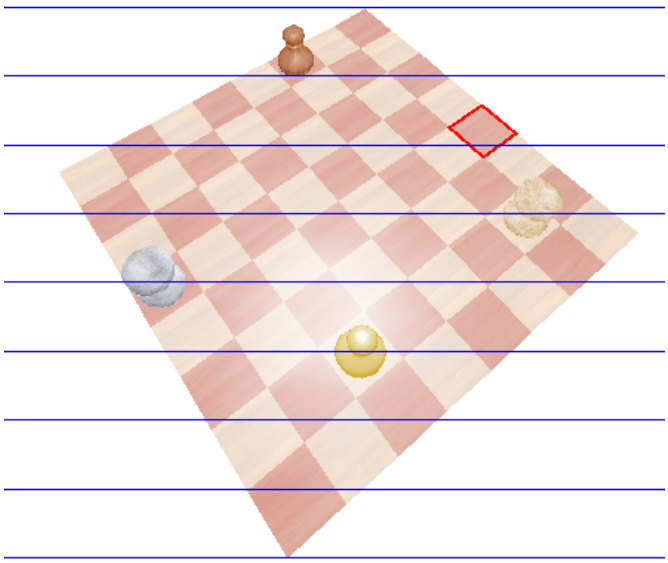

Рис. 4. Принцип работы распараллеленного алгоритма, использующего z-буфер

В качестве вспомогательного этапа целесообразно было использовать первый этап алгоритма Робертса, т.е. удаление плоскостей, экранируемых самим телом. Суть этого этапа заключается в том, что удаление нелицевых граней происходит согласно знаку скалярного произведения внешней нормали плоскости и вектора наблюдения. Такой подход значительно увеличивает производительность программы, т.к. операция скалярного произведения выполняется быстро и в случае выпуклых фигур метод отбрасывать все нелицевые грани.

В конечном итоге в работе использовался собственный алгоритм на основе распараллеленного алгоритма использующего z-буфер и алгоритма Робертса.

3.2. Алгоритм заливки треугольника

При выборе алгоритма растеризации треугольника [1] следует руководствоваться в первую очередь тем, что в процессе растеризации происходит билинейная интерполяция геометрических координат, интенсивности освещения и текстурных координат. Исходя из этого, в проекте был использован алгоритмы описанный на рис. 5. Описанный алгоритм является общим случаем, его многопоточная реализация с билинейной интерполяцией множества параметров не приводится в силу отсутствия фундаментальных различий.

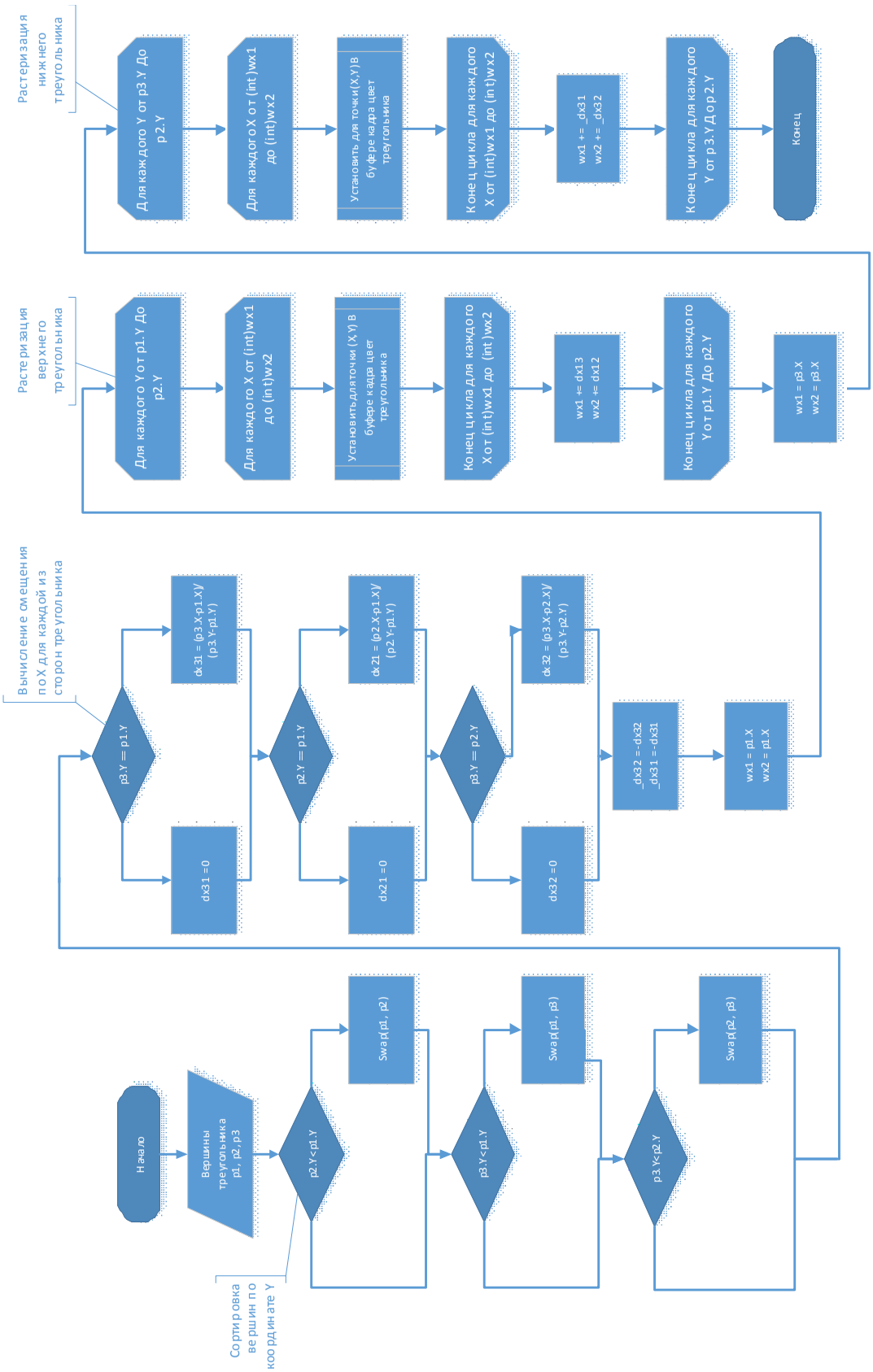

Рис. 5. Схема алгоритма заливки треугольника

3.3. Механизм изменения интенсивности цвета, заданного в цветовой схеме RGB

Одна из проблем, вставших при разработке программы моделирования, заключалась в том, что в силу наличия текстур появилась необходимость в механизме изменения интенсивности любого пикселя, цвет которого задан в цветовой схеме RGB, таким образом, чтобы цвет при минимальной интенсивности становился черным, а при максимальной – белым. Отсутствие текстур позволило бы решить задачу изменения интенсивности освещения тривиальным образом при билинейной интерполяции цвета каждой из вершин.

Для конвертации из и в RGB наилучшим образом подошла цветовая схема Hue/Saturation/Lightness [3] (далее HSL), где компонента L – яркость, принимающая значения от 0 до 1. Изменяя ее, можно достичь необходимого результата.

Наличие проблемы наложения текстуры вызвало необходимость решить эту задачу следующим образом.

А) В процессе растровой развертки треугольника, принадлежащего фигуре, произвести билинейную интерполяцию величины яркости и текстурных координат.

Б) Для каждого пикселя получить цвет RGB по текстурным координатам и сконвертировать полученный цвет в HSL.

В) Задать поле яркости в HSL и сконвертировать обратно в RGB.

Г) Занести измененный цвет RGB в буфер кадра.

4. Последовательность видовых преобразований

В ходе анализа и выбора алгоритмов трехмерной графики была выбрана последовательность преобразований, приведенная на рис. 6:

Рис. 6. Общая последовательность видовых преобразований

5. Приемы оптимизации программного кода

В разработанной программе применялось множество приемов оптимизации кода, среди которых

a) алгоритмические методы оптимизации: распараллеливание алгоритма, использующего z-буфер, использование специализированного под билинейную интерполяцию алгоритма растеризации треугольника, предварительная фильтрация нелицевых граней первых этапом алгоритма Робертса;

b) оптимизация средствами языка.

Так как в качестве языка программирования при разработке программы был выбран язык C#, а программы, написанные на нем, значительно уступают в производительности аналогичным, написанным на таких языках как C/C++ или Rust, пришлось помимо алгоритмических методов оптимизации кода использовать оптимизацию средствами языка.

Рассмотрим несколько приемов оптимизации кода на языке C# и использованных при разработке проекта.

1) Использование структур вместо классов там, где это возможно. Известно, что доступ к памяти и извлечение из памяти является наиболее затратной операцией. В то же время в языке C# существует 2 основные структуры данных: класс (class) и структура (struct). Второй отличается от первого в плане производительности в первую очередь тем, что в структуре нельзя использовать наследование и объекты структуры хранятся в памяти единым блоком. Также известно, что скорость доступа к полям структуры в среднем в 20 раз выше, чем аналогичный доступ к полям класса. Примером использования структур в программе является структура конвертации из RGB в HSL и обратно.

2) Запрет на foreach. Причиной столь резкой критики является тот факт, что любой foreach создает объект интерфейса IEnumerable, что куда хуже, чем обычный for, который дизассемблируется в пару строк ассемблерного кода.

3) Ограничение использования указателей. Безусловно, unsafe код значительно увеличивает производительность программы, но необходимо понимать, что использование любого указателя на объект, вызывает обращение к Garbage Collector с целью блокировки этого объекта, чтобы изъять данный объект из под контроля сборщика мусора на время существования указателя. Поэтому частое использование указателей приведет к частому обращению к GC на изъятие и возврат объектов под контроль сборщика мусора. В программе использовался указатель на буферы кадра и z-буфер, т.к. требуется всего 4 вызова к GC за один кадр. Также важно заметить, что использование

указателей на буферы приводит к тому, что отсутствуют проверки на выход указателя за границы буферов, представленных в виде одномерных массивов. С одной стороны, это хорошо, т.к. не происходит проверки на выход за границы массива при каждом запросе к элементу массива, но, с другой стороны, неосторожное использование указателей может привести к повреждению других участков памяти.

4) Inline функции. По умолчанию в .Net static и private методы являются inline. Однако долгое время у программистов отсутствовал механизм назначения метода inline (встроенным). Начиная с .Net 4.5 появился атрибут `MethodImplOptions.AggressiveInlining`, который настоятельно рекомендует компилятору назначить метод, содержащий данный атрибут, встроенным.

Тестирование ПО

Тестирование данной программы проводилось на компьютере на базе процессора Intel Core i5 с тактовой частотой 2300 МГЦ. При этом загрузка процессора составляла от 25 до 80%.

На рис. 7 изображен график зависимости FPS от количества потоков, используемых для генерации кадра. Количество треугольников – 7328.

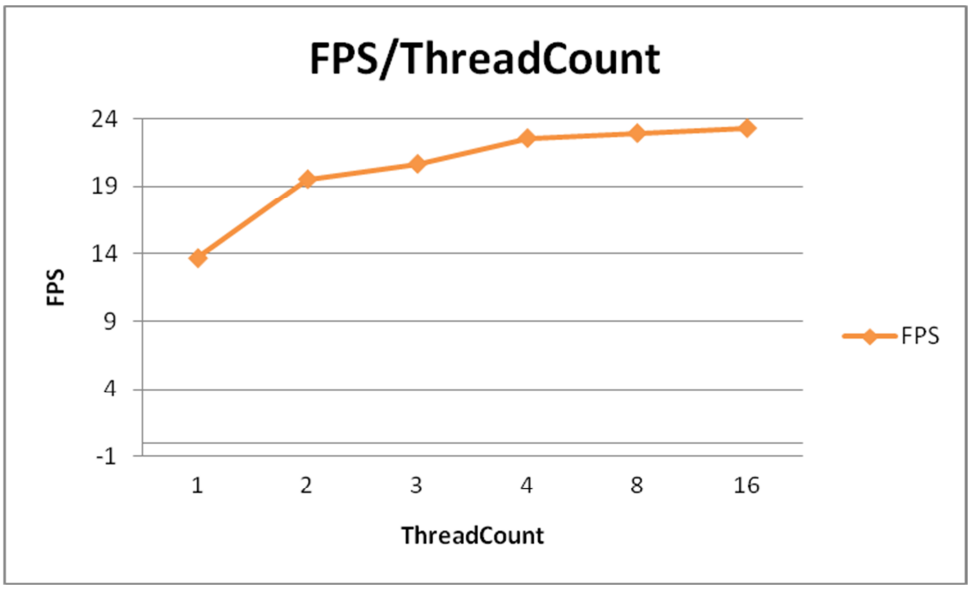

Рис. 7. Тест производительности

Заключение

В работе были рассмотрены основные алгоритмы компьютерной графики и их модификации, использованные при разработке программы моделирования фигур вращения, в частности:

- предложены математические методы создания модели фигуры вращения;

- описан разработанный в ходе реализации программы универсальный алгоритм расчета текстурных координат для фигур вращения;
- был предложен новый алгоритм удаления невидимых линий, состоящий из первого этапа алгоритма Робертса и распараллеленного варианта алгоритма, использующего z-буфер;
- рассмотрен механизм изменения цветовой схемы с учетом изменения интенсивности;
- предложена общая последовательность видовых преобразований;

Описаны использованные в работе приемы оптимизации программного кода, написанного на языке C#.

Были приведены результаты тестирования, на основании которых можно сделать выводы о том, что использование параллельной реализации z-буфера дает большой выигрыш при использовании всех четырех виртуальных ядер процессора, на котором тестировалась программа.

Список литературы

1. Алгоритм растеризации треугольника. Режим доступа: http://compgraphics.info/2D/triangle_rasterization.php (дата обращения 13.08.2014).
2. Захаров А.А. Математические модели геометрических объектов. Методы интерполяции кубическими сплайнами. Режим доступа: http://www.bmstu.ru/ps/~azaharov/fileman/download/%D0%93%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5%20%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D0%B8/2.pdf (дата обращения: 20.10.2014).
3. Wikipedia. Hue-Saturation-Lightness – Wikipedia, The Free Encyclopedia. 2014. Режим доступа: https://ru.wikipedia.org/wiki/HSL. (дата обращения: 20.10.2014).
4. Wikipedia. Bezier curve – Wikipedia, The Free Encyclopedia. 2014. Режим доступа: https://en.wikipedia.org/wiki/B%C3%A9zier_curve (дата обращения 21.07.2014).
5. Wikipedia. UV mapping – Wikipedia, The Free Encyclopedia. 2014. Режим доступа: https://en.wikipedia.org/wiki/UV_mapping (дата обращения 27.07.2014).
6. Роджерс Д.Ф. Алгоритмические основы машинной графики. М.: Мир, 1989. 503 с.