

Федеральное агентство по образованию
ГОУ ВПО «Волгоградский государственный университет»
факультет Математики и информационных технологий
кафедра Информационных систем и компьютерного моделирования

Н. М. Кузьмин, Е. В. Верстаков

КОМПЬЮТЕРНАЯ ГРАФИКА И МОДЕЛИРОВАНИЕ

Учебное пособие

Волгоград 2010

УДК 004.92
ББК 681.3.07

Рецензент:

Составители:
Н. М. Кузьмин,
Е. В. Верстаков

Рекомендовано к печати ученым советом факультета
Математики и информационных технологий ВолГУ
(протокол № 6 от 25.01.2010 г.)
и учебно-методической комиссией
по блоку образовательных программ «Информационные технологии».

Компьютерная графика и моделирование: Учебное пособие / сост.: Н. М. Кузьмин, Е. В. Верстаков. — Волгоград: Волгоградское научное издательство. — 2010. — 120 с. — 50 экз.

ISBN

В данном учебном пособии описан лабораторный практикум по курсам «Компьютерная графика» и «Компьютерная геометрия и графика» для студентов направления подготовки бакалавров 230100 — «Информатика и вычислительная техника», специальностей 230200 — «Информационные системы и технологии», 220200 — «Автоматизированные системы обработки информации и управления» и других технических и математических специальностей и направлений. Лабораторный практикум составлен в соответствии с требованиями Государственного образовательного стандарта высшего профессионального образования.

УДК 004.92
ББК 681.3.07

ISBN

Содержание

1	Растровый графический редактор GIMP	4
2	Векторный графический редактор Inkscape	15
3	Трёхмерное геометрическое моделирование в пакете Google SketchUp	25
4	Основы OpenGL	35
5	Формат хранения графической информации BMP	53
6	Обработка растровых изображений	60
7	Растровые преобразования	66
8	Алгоритмы удаления невидимых линий и поверхностей	83
9	Алгоритмы закраски	108
	Литература	118

1 Растровый графический редактор GIMP

Цель работы

Целью выполнения данной работы является изучение основных понятий растровой графики и получение навыков работы в растровом графическом редакторе.

Теоретические сведения

Растровое изображение — это файл данных или структура, представляющая собой сетку пикселей или точек цветов (обычно прямоугольную) на компьютерном мониторе, бумаге и других отображающих устройствах и материалах.

Работу в растровом графическом редакторе будем рассматривать на примере пакета GIMP (<http://www.gimp.org>). Это свободно распространяемый и динамично развивающийся программный продукт для работы с изображениями в растровом формате. Внешний вид приложения показан на рис. 1.

Элементами окон являются:

1. **Панель инструментов:** в ней содержится главное меню, кнопки с пиктограммами, с помощью которых производится выбор инструментов, и некоторые другие полезные вещи.
2. **Параметры инструментов:** под панелью инструментов прикреплен диалог «Параметры инструментов», который отображает параметры выбранного инструмента (в данном случае это инструмент «Кисть»).
3. **Окно изображения:** каждое изображение в GIMP отображается в отдельном окне. Вы можете открыть одновременно достаточно большое количество изображений, столько, сколько позволяют системные ресурсы.
4. **Диалог «Слои/Каналы/Контур/Отменить»:** этот диалог отображает структуру слоев активного изображения и позволяет управлять

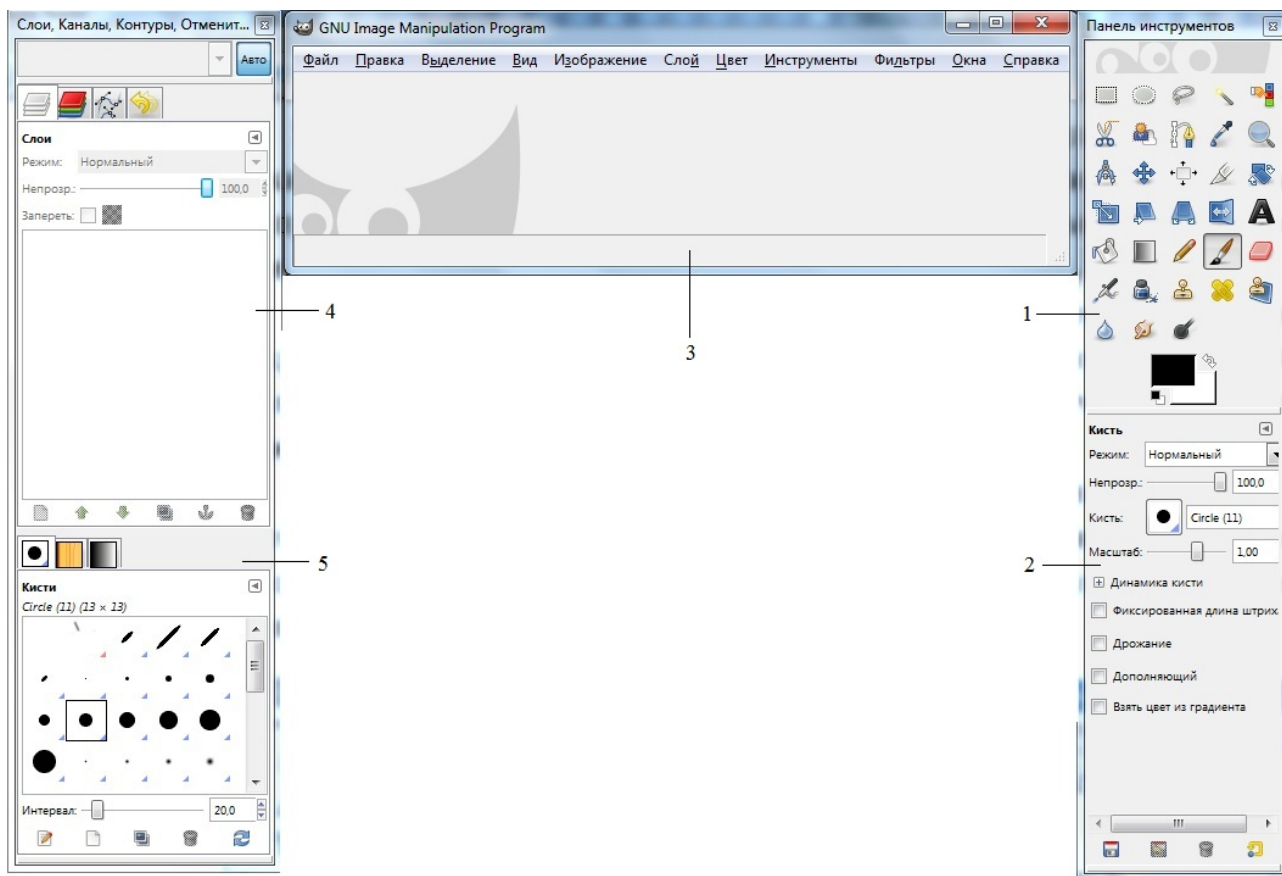


Рис. 1: Внешний вид растрового редактора GIMP.

ими.

5. **Кисти/Текстуры/Градиенты:** панель, расположенная ниже диалога слоев, показывает диалоги управления кистями, текстурами и градиентами.

Приведенный набор — это минимальный набор окон. В GIMP используется более дюжины различных диалогов, которые можно открыть при необходимости. Опытные пользователи обычно держат открытыми панель инструментов (с параметрами инструментов) и диалог «Слои».

При работе с многослойным изображением диалог «Слои» необходим всегда. В отличие от многих других программ, в GIMP нет возможности разместить все содержимое — панели и диалоги — в одном цельном окне.



Изображение — основной объект, с которым работает GIMP. Под словом «изображение» подразумевается один файл. Можно отождествлять изобра-



жение и окно, которое его содержит, но это будет не совсем правильно: можно открыть несколько окон с одним и тем же изображением. В то же время нельзя открыть в одном окне более одного изображения, и нельзя работать с изображением без отображающего его окна. Изображение в GIMP может быть достаточно сложным. Наиболее правильной аналогией будет не лист бумаги, а, скорее, книга, страницы которой называются слоями.

Если изображение подобно книге, то слой можно сравнить со страницей внутри книги. Простейшее изображение содержит только один слой и, продолжая аналогию, является листом бумаги. Слои могут быть прозрачными и могут покрывать не все пространство изображения.

В GIMP каналы являются наименьшей единицей подразделения стека слоев, из которых создается изображение. Каждый канал имеет тот же размер, что и слой, и состоит из тех же пикселей. Смысл этого значения зависит от типа канала, например, в цветовой модели RGB значение канала R означает количество красного цвета, добавляемого к другим цветам пикселей.

Часто при работе возникает необходимость изменить только часть изображения. Для этого существует механизм выделения областей. В каждом изображении можно создать выделенную область, которая, как правило, отображается в виде движущейся пунктирной линии (она также называется «муравьиной дорожкой»).

Почти все, что делается с изображением, может быть отменено. Вы можете отменить последнее действие, выбрав в меню изображения «Правка → Отменить», но эта операция применяется так часто, что лучше запомнить сочетание клавиш  + .

Сама отмена также может быть отменена. После отмены действия можно вернуть его, выбрав в меню изображения пункт «Правка → Повторить» или с использованием клавиш быстрого доступа  + . Часто это полезно при оценке эффекта какого-либо действия, с помощью его неоднократной отмены и повтора.

Если Вы часто используете отмену и возврат на множество шагов за раз, возможно будет более удобно работать с диалогом «Истории отмен» — при-

крепляемой панелью, которая показывает небольшие эскизы каждой точки в истории отмены, позволяя перемещаться назад или вперед к точке, по которой вы щелкаете.

Типы растровых изображений

Изображение в GIMP — это сложная структура, которая содержит множество составляющих: слои, маски выделения, набор каналов, набор контуров, историю отмен и т. д.

Основное свойство изображения — это режим. Существует три доступных режима: RGB, градации серого и индексированное.

Цветовая модель RGB наиболее проста для понимания и очевидна. В этой модели работают мониторы и бытовые телевизоры. Любой цвет считается состоящим из трех основных компонентов: красного (Red), зеленого (Green) и синего (Blue). Эти цвета называются основными. Считается также, что при наложении одного компонента на другой яркость суммарного цвета увеличивается. Совмещение трех компонентов дает нейтральный цвет (серый), который при большой яркости стремится к белому цвету.

Метод получения нового оттенка суммированием яркостей составляющих компонентов называют аддитивным методом. Он применяется всюду, где цветное изображение рассматривается в проходящем свете («на просвет»): в мониторах, слайд-проекторах и т. п. Нетрудно догадаться, что чем меньше яркость, тем темнее оттенок. Поэтому в аддитивной модели центральная точка, имеющая нулевые значения компонентов (0, 0, 0), имеет черный цвет (отсутствие свечения экрана монитора). Белому цвету соответствуют максимальные значения составляющих (255, 255, 255). Модель RGB является аддитивной, а ее компоненты — красный, зеленый и синий — называют основными цветами.

В изображении в режиме градаций серого, каждая точка представлена уровнем яркости в диапазоне от 0 (черный) до 255 (белый), с набором промежуточных значений, которые представляют различные оттенки серого цвета. На самом деле, и RGB и серые изображения имеют дополнительный цветовой

канал, называемый альфа-каналом, представляющий непрозрачность. Когда значение альфа в данной точке равно нулю, слой совершенно прозрачен и цвет в том месте определяется цветом слоя, лежащего ниже. Когда значение альфа максимально, слой непрозрачен и цвет определяется цветом слоя. Промежуточные значения альфа соответствуют разным степеням полупрозрачности: цвет в выбранной точке изображения — пропорциональная смесь цветов данного слоя и слоя, расположенного ниже.

Третий тип изображений — это индексированные изображения. В индексированном изображении используется ограниченный набор цветов, обычно не более 256. Эти цвета формируют цветовую карту изображения, и каждой точке в изображении назначается цвет из цветовой карты.

Некоторые наиболее используемые типы файлов (включая GIF и PNG) при открытии их в GIMP выводят индексированные изображения. Многие инструменты GIMP работают с индексированными изображениями некорректно, поэтому перед работой с изображением лучше преобразовать его в режим RGB. Если необходимо, вы можете преобразовать его обратно в индексированный режим перед сохранением.

Создание и открытие файлов


В GIMP можно создать новое изображение при помощи пункта меню «Файл → Новый». При этом откроется диалог «Создать новое изображение», где можно установить начальные ширину и высоту.


Открыть уже существующий файл можно при помощи пункта меню «Файл → Открыть». При этом откроется стандартный диалог открытия файлов.



Геометрические фигуры

Создание фигур. Создать и залить геометрические фигуры такие, как квадрат, прямоугольник, круг или эллипс, Вам помогут пункты описанные ниже:

1. Выберите инструмент «Выделение прямоугольных областей».

2. Проведите, зажав левую кнопку мыши, по изображению, чтобы выделить прямоугольную область.
3. Если Вы зажмете кнопку  при выделении, у Вас получится квадратная выделенная область.
4. Теперь, выберите инструмент «Заливка цветом или шаблоном».
5. Выберите цвет переднего плана, в диалоге инструментов GIMP, и значение опции «Тип заливки» поставьте в «Заливка цветом переднего плана», в диалоге инструмента «Заливка цветом или шаблоном».
6. Кликнув левой кнопкой мыши внутри выделенной области, Вы создадите квадрат или прямоугольник.

Проделайте эти же действия, используя инструмент «Выделение эллиптических областей», для создания эллипса или круга (зажав кнопку  при выделении).


Границы фигур. Следующим нашим шагом будет создание границы фигуры. Создание границы должно производиться только после выполнения всех инструкций предыдущего раздела. Удостоверьтесь, что Вы выполнили все инструкции и, что выделенная область до сих пор активна (выделенная область показывается пунктирной линией). Кликните правой кнопкой мыши на изображении, и в появившемся меню выберите в меню изображения «Выделение → Уменьшение». Введите количество пикселей, на которое Вы хотите уменьшить выделенную область. Наконец, очистите («Правка → Очистить» или  + ) уменьшенную выделенную область.

Эта техника может быть применена к любой выделенной области.

Многоугольники. Рассмотрим решение для создания простых фигур — инструмент «Создание и редактирование контуров».

1. Выберите инструмент «Создание и редактирование контуров».

2. Включите опцию «Показывать сетку» («Меню изображения → Просмотр → Показывать сетку»).
3. Используя меню изображения «Изображение → Настроить сетку», Вы можете настроить отображение сетки по своему вкусу. В зависимости от того, какого размера Вы хотите сетку, Вы можете изменять расстояние между точками.
4. Для того, чтобы сетка приносила пользу, при создании геометрических фигур — выберите опцию «Выравнивание по сетке» («Просмотр → Выравнивание по сетке»).

Используя инструмент «Создание и редактирование контуров» кликните левой кнопкой мыши в верхнем левом углу изображения, и, пропустив три ячейки, ниже. Для того, чтобы закрыть контур зажмите кнопку  и кликните на первой точке контура. Чтобы сделать границу контура, используйте встроенные функции обведения GIMP. Эти функции доступны в диалоге инструмента контуров, перейдите в него и нажмите кнопку «Обвести по контуру». Появится окно, в котором Вы можете определить опции обведения. Вы можете свободно экспериментировать с опциями обведения для получения нужного эффекта.

Если же Вы хотите залить получившуюся с помощью контура фигуру, нажмите кнопку «Создать выделенную область из контура» в диалоге опций инструмента контуров. Получится фигура, и Вы можете ее залить, используя инструмент «Заливка цветом или шаблоном».

Выделение

Часто при операциях на изображении вам необходимо чтобы изменения затронули лишь его часть. В GIMP вы делаете это с помощью выделения этой части. Каждое изображение имеет ассоциированное с ним выделение. Большинство (но не все) операций в GIMP применяются только к выделенным частям изображения.

Существует много, множество ситуаций, где всего лишь создание правильного выделения является ключом к получению желаемого результата, и часто это не очень просто сделать. К примеру, когда на изображении необходимо вырезать некоторый объект с фона, и вставить его в другое изображение. Для того, чтобы это сделать, нужно создать выделение, которое содержит интересующий нас объект, и ничего кроме него. Это зачастую трудно, поскольку ранние объекты могут иметь очень сложную форму, и в некоторых местах их трудно отличить от расположенных позади объектов.

Обычно когда вы создаёте выделение, вы видите его как пунктирную линию, которая окружает часть изображения. Выделения представляет собой разновидность контейнера, с выделенными частями внутри, и невыделенными частями снаружи изображения. Эта концепция выделения является подходящей в большинстве случаев, но в действительности она не совсем корректна.

В действительности выделение реализовано в виде канала. В терминах этой внутренней структуры он идентичен красному, зеленому, синему, и альфа каналам изображения. Следовательно, выделение содержит определённое к каждому пикселу изображения значение, в диапазоне между 0 (не выделен) и 255 (полностью выделен). Преимущество данного подхода заключается в возможности частичного выделения некоторых пикселей, устанавливая для них значения между 0 и 255. Существует множество ситуаций, где желательно иметь сглаженные переходы между выделенными и невыделенными областями.

Пунктирная линия, появляющаяся когда вы создаёте выделение — контурная линия, разделяющая выделенные более чем наполовину области от областей, которые выделены меньше чем наполовину.

Кисти

Кисть — пиксельное изображение или набор пиксельных изображений, используемых GIMP для рисования. GIMP включает в себя набор из 10 инструментов рисования, которые предоставляют не только операции, непосред-

ственно связанные с рисованием, но и такие функции, как стирание, копирование, размытие, освещение, затемнение и т. д. Все инструменты рисования, за исключением пера, используют один и тот же набор кистей. Пиксельное изображение кисти соответствует отпечатку, сделанному одиночным «касанием» кисти изображения («один клик»). Мазок кистью обычно создается движением курсора по изображению с нажатой кнопкой мыши. Таким образом создается серия отпечатков по указанной траектории, методом, определенным характеристиками кисти и используемым инструментом рисования.

Кисти могут быть выбраны щелчком по пиктограмме в диалоге выбора кистей. Выбранная кисть отображается в области «Кисти/Шаблоны/Градиенты» панели инструментов. Один из вариантов вызова диалога кистей это щелчок по пиктограмме кисти в этой области.

В базовой установке GIMP есть небольшое количество кистей. Некоторые из них довольно причудливы и вряд ли могут быть использованы по прямому назначению. Такие кисти существуют в наборе для того, чтобы дать представление о возможностях создания кистей. Новые кисти можно создать самостоятельно, а можно скачать уже готовые.

GIMP использует несколько различных типов кистей. Все они применяются одинаково, и, в большинстве случаев, не важно, каким типом кисти Вы рисуете.

Градиенты

Градиент представляет собой набор расположенных в линейной последовательности цветов. В основном градиенты применяются инструментом «Заливка», также известным как «Градиент» или «Заливка градиентом»: он заливает выделение цветами из градиента. Для контроля размещения градиентных цветов внутри выделения Вы можете изменять множество параметров.

Рисование градиентом. Любой из основных инструментов рисования в GIMP даёт Вам возможность использовать цвета из градиента. Это позволяет создавать мазки кистью, которые меняют цвет от одного конца к другому.

Фильтр «Отображение градиента». Этот фильтр позволяет сделать «цветным» чёрно-белое изображение, заменяя каждый оттенок серого соответствующим цветом из градиента.

Вместе с GIMP устанавливается большое количество интересных градиентов, и Вы можете добавлять новые, создавая собственные или загружая из других источников. Для доступа к полному набору доступных градиентов используйте диалог «Градиенты», который вы можете активировать при необходимости, или оставить рядом как закладку в панели. «Текущий градиент», используемый в большинстве операций с градиентом, отображается в области «Кисть/Шаблон/Градиент» панели инструментов. Щелчок по символу градиента в панели инструментов это альтернативный метод вызова диалога градиентов.

Задание

По указанию преподавателя создать растровый рисунок, используя изученные теоретические сведения. Рисунок должен состоять из нескольких слоев.

Контрольные вопросы

1. Что такое растровая графика?
2. Что такое слои и каналы в GIMP?
3. Перечислите типы растровых изображений.
4. Как можно с помощью инструмента «Прямоугольное выделение» создать квадрат?
5. Как можно создать многоугольники?
6. Что такое кисти?
7. Что такое градиент?

Литература

1. GNU Image Manipulation Program. Руководство пользователя [Электронный ресурс]. — GNU Image Manipulation Program, 2010. — Режим доступа: <http://docs.gimp.org/2.6/ru/>. — Загл. с экрана.
2. Шикин, Е. В., Боресков, А. В. Компьютерная графика. Полигональные модели [Текст]. — М.: ДИАЛОГ-МИФИ, 2001. — 464 с.

2 Векторный графический редактор Inkscape

Цель работы





Целью выполнения данной работы является изучение основных понятий векторной графики и получение навыков работы в векторном графическом редакторе.

Теоретические сведения

Векторная графика основана на использовании геометрических примитивов, таких как точки, линии, сплайны и многоугольники, для представления изображений. Термин используется в противоположность к растровой графике, которая представляет изображения как матрицу пикселей (точек).

Работу в векторном графическом редакторе будем рассматривать на примере пакета Inkscape (<http://www.inkscape.org>). Это свободно распространяемый и динамично развивающийся программный продукт для работы с изображениями в векторном формате. Внешний вид приложения показан на рис. 2.

Перемещение по холсту

Для перемещения по холсту с помощью клавиатуры можно использовать комбинацию клавиш  + стрелки. Также передвигаться по холсту можно, зажав его поверхность средней клавишей мыши или при помощи ползунков (для того, чтобы показать или спрятать их необходимо нажать  + ). Колесико прокрутки на мышке также можно использовать для вертикального перемещения. Для горизонтального перемещения можно использовать клавишу  вместе с колесиком.

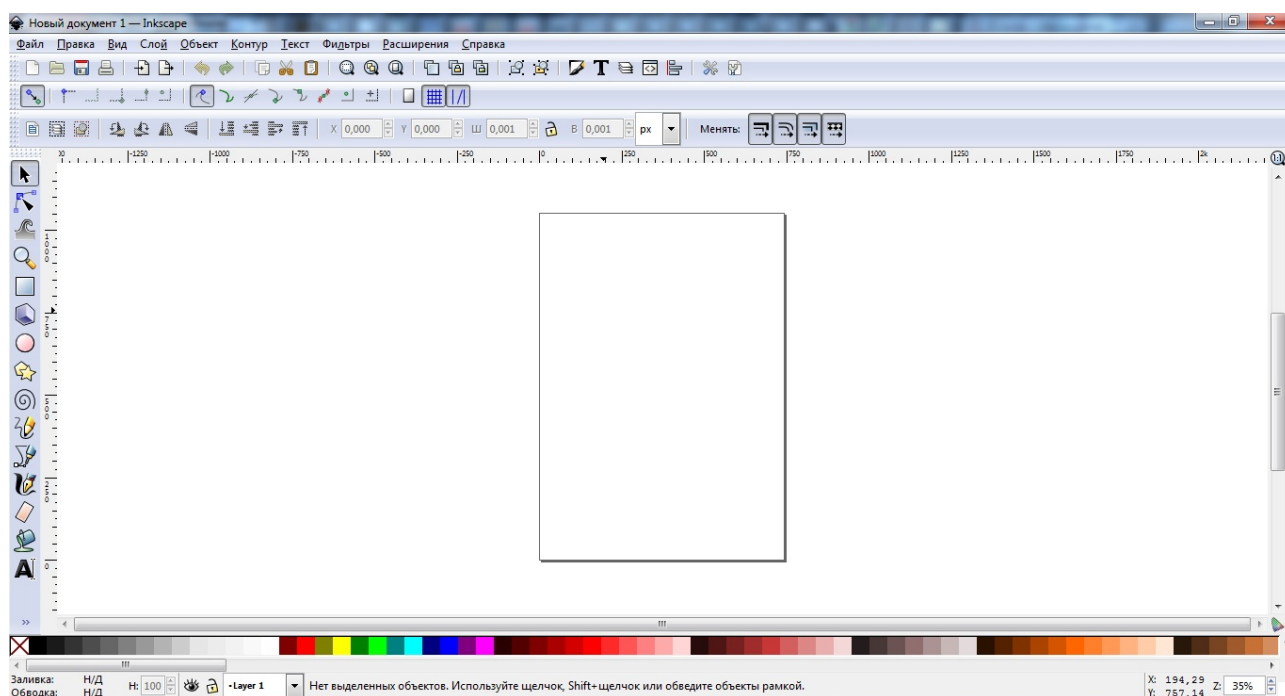














Рис. 2: Внешний вид векторного редактора Inkscape.

Изменение масштаба

Простейший способ изменить масштаб — нажать клавишу  или  (для увеличения работает и клавиша ). Также можно использовать  + средняя клавиша мыши или  + правая кнопка мыши — для увеличения,  + средняя или  + правая — для уменьшения, или колесико мыши с нажатым . Кроме того, можно выбрать масштаб увеличения в нижнем левом углу окна документа. Значение масштаба указано в процентах; набрав нужное, нажмите . Наконец, в программе есть инструмент масштабирования (среди инструментов слева), который позволяет увеличивать только необходимую выделенную область.










Inkscape хранит историю масштабов, которые использовались при работе. Чтобы вернуться к предыдущему состоянию масштаба, нажмите клавишу  или  +  для перехода к следующему.

Инструменты Inkscape



Панель со значками в левой части окна представляет инструменты Inkscape для рисования и редактирования. В верхней части окна, под меню, находится панель управления с основными командными кнопками, и чуть ниже — панель «Параметры инструментов», содержащая параметры, специфичные для каждого инструмента. Строка состояния, в самом низу окна, показывает полезные подсказки во время работы.

Многие действия доступны с клавиатуры. Полный справочник по клавишам можно вызвать через меню «Справка → Использование клавиатуры и мыши».


Работа с документами

Для создания нового чистого документа используйте «Файл → Создать» или нажмите  + . Чтобы открыть существующий документ SVG, используйте «Файл → Открыть» ( + ). Для сохранения используйте «Файл → Сохранить» ( + ), либо «Сохранить как...» ( +  + ) для сохранения файла под другим именем.

Inkscape использует формат SVG (Scalable Vector Graphics — масштабируемая векторная графика) для своих файлов. SVG является открытым стандартом и широко используется в графических пакетах. Формат SVG использует язык разметки XML, поэтому файлы в этом формате могут редактироваться любым текстовым или XML-редактором (отдельно от Inkscape). Помимо SVG в Inkscape можно работать и с другими форматами (например, EPS и PNG).






Для каждого документа Inkscape открывает новое окно. Вы можете переключаться между ними разными способами в зависимости от менеджера окон (например, нажав  +  для переключения между документами по кругу).


Создание фигур

Для создания прямоугольника выберите синий прямоугольник в полоске слева (или нажмите ). Наведите курсор мыши на документ (тут же или в новом созданном окне), нажмите левую клавишу мыши и переместите ее курсор в сторону — должен получиться прямоугольник.



По умолчанию прямоугольник залит синим цветом, имеет черную обводку и частично прозрачен. Ниже будет показано, как изменить эти параметры. Другими инструментами можно создавать овалы, звезды и спирали.

Каждая созданная фигура имеет один или несколько белых прямоугольников управления (ручек). При их перемещении будут происходить изменения параметров фигуры (белые точки видны только когда выбран один из четырех инструментов: синий квадрат, коричневый круг, желтая звездочка или спираль). У панели параметров инструментов свой способ изменения фигуры. Управляющие элементы в ней влияют на выбранные в настоящий момент объекты (т. е. те, ручки которых видны), а также определяют параметры новых фигур.



Для отмены последнего действия действует комбинация  + . (Если вы изменили решение, можно вернуть отменённое действие, используя  +  + .)

Один из наиболее полезных инструментов в Inkscape — «Селектор». Выбрать его можно щелчком по черной стрелке (либо нажатием  или пробела). Этим инструментом можно выбрать любой объект на холсте.

При выборе объекта вокруг него появляются восемь стрелок. После этого можно:

- Передвигать объект (с нажатым  перемещения ограничиваются двумя осями: горизонтальной и вертикальной).
- Менять размер объекта, потянув за любую из стрелок (меняя размер с нажатым , сохраняются пропорции оригинала).











Если щелкнуть по объекту еще раз — направление стрелок изменится. После этого можно:




- Поворачивать объект, потянув за угловые стрелки (с нажатым  объект будет поворачиваться шагами по 15°. Сместив крестик, вы сместите центр вращения.)
- Перекашивать (наклонять) объект, двигая неугловые стрелки (с нажатым  перекашивание будет производиться с шагом в 15°).

В этом режиме (режиме выделения объектов) также можно менять размеры и расположение выделения на холсте, используя поля сверху.

Изменение формы при помощи клавиш



Одна из особенностей Inkscape, отличающая его от большинства других редакторов векторной графики — удобное управление с клавиатуры. Трудно найти команду или действие, которое нельзя было бы выполнить с клавиатуры, и изменение формы объектов — не исключение.





Можно использовать клавиатуру для перемещения объектов (используя стрелки), изменения размера (клавиши  и ), и вращения (клавиши  и ). По умолчанию перемещение и размер меняются на 2 пиксела. С нажатым  это значение увеличивается в 10 раз (и становится равным 20 пикселям). Клавиши  +  и  +  увеличивают или уменьшают объект на 200% или 50% от оригинала соответственно. С нажатым  вращение будет выполняться с шагом в 90° вместо 15°.

Удобными являются пиксельные изменения формы, производимые с нажатой клавишей  и клавишами изменения форм. Например,  + стрелки будут двигать выбранное на 1 пиксел данного масштаба (т.е. на 1 пиксел экрана, не путайте с пикселом, который является SVG единицей длины и отличается от пиксела масштаба). Это значит, что если масштаб был увеличен, то  + стрелка даст меньшее смещение от абсолютного измерения, что по-прежнему будет выглядеть как смещение на пиксел на экране. Это даёт возможность точно разместить объект, меняя масштаб.

Схожим образом  +  и  +  меняют размер на один пиксел, а  +  и  +  вращают объект на один пиксел.









Выделение нескольких объектов




Есть возможность выбрать любое количество объектов одновременно, нажав  + щелчок на желаемых объектах. Еще можно выбрать объекты рамкой выделения — так называемым «резиновым» выделением. (Рамка выделения появляется, когда выделение начинается с пустого места, а с нажатой клавишей  рамка выделения появится и над объектом.) Каждый выделенный объект отображается с пунктирной рамкой вокруг него.

 + щелчок на выделенном исключает его из общего выделения. Нажатие  сбросит все выделения.  +  выделяет все объекты в пределах активного слоя (если слои не создавались, это равносильно выделению всех объектов документа).

Группировка




Несколько объектов могут быть объединены в группу. При перемещении и трансформации группа ведет себя как обычный объект.

Для создания группы нужно выбрать один или более объектов и нажать  + . Разгруппировать их можно, нажав  +  и предварительно выбрав группу. Сами по себе группы могут быть сгруппированы и как одиночные объекты. Подобная поэтапная группировка может быть сколь угодно сложной. При этом следует помнить, что  +  разгруппирует только последнюю группировку. Нужно нажать  +  несколько раз, если необходимо полностью разгруппировать сложно сгруппированные группы в группе.

Очень удобно, что не нужно разбивать группу для редактирования отдельных объектов. Выбрать объект можно выполнив  + щелчок по объекту, после чего его можно редактировать. Таким же образом работает комбинация  +  + щелчок, позволяющая редактировать несколько объектов независимо от группы.

Заливка и обводка

Множество функций Inkscape доступны через диалоги (подменю). Вероятно, самый простой способ заполнить объект каким-либо цветом — это выбрать «Образцы цветов...» из меню «Вид», выбрать объект и его цвет в палитре образцов цвета (изменение цвета заливки или обводки объекта).

Более мощным способом является использование диалога «Заливка и обводка...» через меню «Объект» ( +  + ). Диалог содержит три вкладки: «Заливка», «Цвет обводки», и «Стиль обводки». Вкладка «Заливка» позволяет изменить заполнение выбранного объекта (или объектов). Используя кнопки под вкладкой, можно выбрать тип заливки, включая режим «Нет заливки» (кнопка со знаком «X»), режим «Плоский цвет», режимы «Линейный градиент» или «Радиальный градиент».



Чуть ниже расположены кнопки-варианты выбора цвета. Каждый вариант имеет свою вкладку: «RGB», «CMYK», «HSL» и «Круг». Вероятно, самым удобным вариантом является «Круг», где можно выбрать тон цвета, вращая треугольник, а затем подобрать насыщенность и яркость в самом треугольнике. Все варианты выбора цвета имеют возможность менять альфа-канал (прозрачность) выбранного объекта (или объектов).

Каждый раз при выборе объекта вкладка «Заливка и обводка...» показывает текущее значение для данного объекта (для нескольких, одновременно выбранных объектов, вкладка цвета показывает их усредненный цвет).

Используя вкладку «Цвет обводки», вы можете убрать обводку объекта, установить его цвет или прозрачность.



Последняя вкладка «Стиль обводки» позволяет изменить толщину и другие параметры обводки.





И, наконец, вместо сплошной окраски можно использовать градиенты для заливки и/или обводки.

Ещё один способ изменить цвет объекта — использовать инструмент «Пипетка» («Брать усредненные цвета от изображения» (). Достаточно просто щелкнуть этим инструментом в любой части рисунка, и полученный цвет будет присвоен выбранному до этого объекту ( + щелчок присвоит




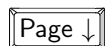
цвет обводке).




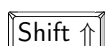

Дублирование, выравнивание, распределение



Одним из наиболее распространенных действий является дублирование объекта ( + ). Дублирование размещает дубликат над оригиналом и делает его выделенным, так что его можно переместить в сторону при помощи мыши или клавиш со стрелками.

Если объекты стоят неровно, это можно поправить, используя диалог «Выровнять» ( +  + ). Выберите все объекты, которые необходимо выровнять ( + щелчок или выделив мышью), откройте диалог (в меню «Объект») и нажмите на кнопку «Центрировать на горизонтальной оси», а после на кнопку «Выровнять интервалы между объектами по горизонтали» (читайте подсказки над кнопками).

Z-порядок






Термин Z-порядок (порядок по оси *Z*) относится к перекрыванию объектами друг друга на рисунке. Иначе говоря, Z-порядок определяет, какой объект находится выше и закрывает собой другие. Две команды в меню «Объект → «Поднять на передний план» (клавиша ) и «Опустить на задний план» (клавиша ), переместят выбранный объект в самую верхнюю или самую нижнюю позицию по оси *Z* данного слоя. Другие две команды, «Поднять» () и «Опустить» (), опустят или приподнимут выбранный объект (или объекты), но только на один уровень относительно других невыделенных объектов по оси *Z* (считаются только объекты, перекрывающие выделенные; если выделение ничем не перекрывается, действие «Поднять» и «Опустить» будет ставить его в самую верхнюю или самую нижнюю позицию соответственно).


Очень полезная клавиша для выделений объектов —  . Если ничего не выбрано,  выделяет самый нижний объект по оси *Z*; иначе  выбирает объект, находящийся над выбранным объектом (объектами) на оси *Z*.  +  срабатывает наоборот, переключая от верхнего к нижнему,

поскольку при создании объекта он добавляется в самый верх Z -уровня. И если нет выделения, нажатие  +  выберет последний созданный объект.

Выделение объектов под объектами и перемещение выделенного

Вы можете видеть нижний объект, если верхний (частично) прозрачен, но щелкнув по нужному, Вы сделаете выделенным верхний объект, а не тот, что нужен.

В такой ситуации может помочь комбинация  + щелчок. Для начала щелкните объект, зажимая при этом клавишу . В результате будет выбран тот объект, что сверху, как и при обычном выделении. Но при повторном нажатии  + щелчок в этом же месте выделенным станет нижний объект, ещё нажатие — и выделение сместится на объект уровнем ниже и т. д. Таким образом, несколько нажатий  + щелчок на стопке объектов будут перемещать выделение от верхнего объекта к нижнему на оси Z . Добравшись до самого нижнего объекта, нажатие  + щелчок выберет самый верхний объект.

Вы можете менять форму самого нижнего объекта и передвигать за управляющие ручки, но при попытке перемещения самого объекта Ваше выделение сбросится и выделенным станет объект, находящийся выше (таким образом работает система щелчок-и-перемещение — сначала она выбирает объект (верхний) под курсором, а потом уже даёт возможность его перемещать). Чтобы указать, что перемещать нужно то, что выбрано сейчас, не выбирая ничего другого, используйте  + перемещение (мышью). Эта комбинация будет перемещать нужное выделение вне зависимости от того места, где движется курсор мыши.

Задание

По указанию преподавателя создать векторный рисунок, используя изученные теоретические сведения.

Контрольные вопросы

1. Что такое векторная графика?
2. Как можно перемещаться по холсту?
3. Какой формат файлов используется в Inkscape как основной?
4. Перечислите базовые фигуры, которые можно создать в Inkscape.
5. Опишите способы изменения параметров фигур.
6. Как выделить несколько объектов сразу? Как создать группу из нескольких объектов?
7. Опишите, как можно менять параметры заливки и обводки фигур.
8. Как продублировать объекты, как выровнять их?
9. Что такое Z-порядок? Как можно работать с объектами, находящимися один над другим?

Литература

1. Inkscape tutorial [Электронный ресурс]. — Inkscape.org, 2009. — Режим доступа: <http://inkscape.org/doc/basic/tutorial-basic.ru.html>. — Загл. с экрана.
2. Шикин, Е. В., Боресков, А. В. Компьютерная графика. Полигональные модели [Текст]. — М.: ДИАЛОГ-МИФИ, 2001. — 464 с.

3 Трёхмерное геометрическое моделирование в пакете Google SketchUp

Цель работы

Теоретические сведения

Трёхмерная графика оперирует с объектами в трёхмерном пространстве. Обычно результаты представляют собой плоскую картинку, проекцию. Трёхмерная компьютерная графика широко используется в кино, компьютерных играх.

Трёхмерное изображение на плоскости отличается от двумерного тем, что включает построение геометрической проекции трёхмерной модели сцены на плоскость (например, экран компьютера) с помощью специализированных программ.

Для получения трёхмерного изображения на плоскости требуются следующие шаги:

- моделирование — создание трёхмерной математической модели сцены и объектов в ней;
- рендеринг (визуализация) — построение проекции в соответствии с выбранной физической моделью;
- вывод полученного изображения на устройство вывода — дисплей или принтер.

Рассматривать трёхмерное геометрическое моделирование будем на примере программного пакета Google SketchUp (<http://sketchup.google.com/>). Это программа для быстрого создания и редактирования трёхмерной графики. Внешний вид программы показан на рис. 3.

По сравнению со многими популярными пакетами обладает рядом преимуществ, заключающихся, в первую очередь, в почти полном отсутствии окон предварительных настроек. Все геометрические характеристики задаются в

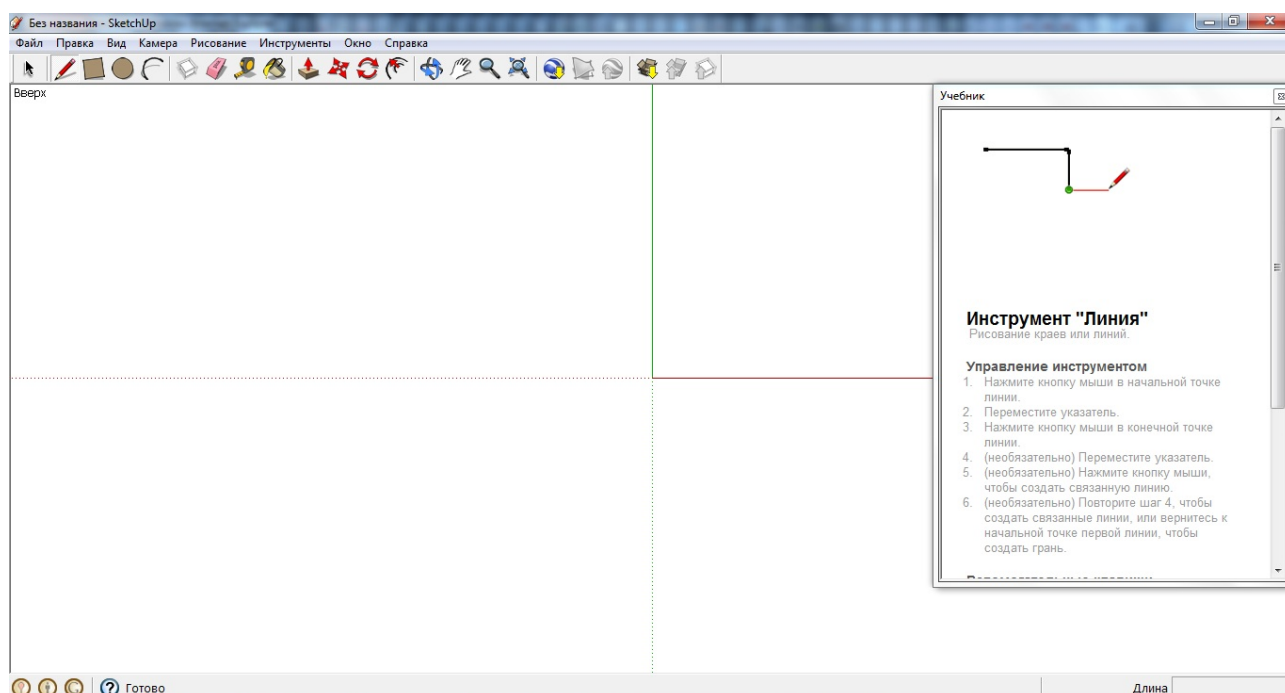


Рис. 3: Внешний вид трехмерного редактора google SketchUp.

процессе построения объекта. Они заносятся с клавиатуры до или сразу после окончания действия инструмента, что отражается в Value Control Box (Панель Контроля Параметров). Эта особенность позволяет избежать необходимости настраивать каждый инструмент перед его применением, а затем редактировать возможные неучтённые ошибки. В то же время, это достоинство оборачивается недостатком, когда возникает потребность в массовом изменении геометрии созданных объектов. Тем не менее, такие ситуации возникают редко, а их исправление средствами пакета не составляет большого труда.

Инструмент «Выбрать»


Осуществляет выбор объектов для изменения при работе с другими инструментами или командами.

Вспомогательные клавиши:

 — добавление объекта к выбранным объектам;

 +  — удаление объекта из выбранных объектов;

 — добавление или удаление объекта из выбранных объектов;

 +  — выбор всех видимых объектов модели.

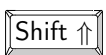
Инструмент «Линия»




Осуществляет рисование краев или линий.

Управление инструментом:

1. Нажмите кнопку мыши в начальной точке линии.
2. Переместите указатель.
3. Нажмите кнопку мыши в конечной точке линии.
4. (необязательно) Переместите указатель.
5. (необязательно) Нажмите кнопку мыши, чтобы создать связанную линию.
6. (необязательно) Повторите шаг 4, чтобы создать связанные линии, или вернитесь к начальной точке первой линии, чтобы создать грань.

Вспомогательные клавиши:

 — блокировать текущее направление логического вывода линии;

 ,  ,  — блокировать определенное направление логического вывода линии (вверх — синяя, влево — зеленая, вправо — красная).

Инструмент «Прямоугольник»


Используется для рисования прямоугольных граней.

Управление инструментом:

1. Нажмите кнопку мыши, чтобы установить первый угол.
2. Переместите указатель по диагонали.
3. Нажмите, чтобы установить второй угол.

4.  — отменить операцию.


Вспомогательные клавиши:

 — блокировать текущее направление логического вывода прямо-
угольника.


Инструмент «Окружность»

Используется для рисования окружностей.

Управление инструментом:

1. Нажмите кнопку мыши, чтобы разместить центральную точку.
2. Переместите указатель от центральной точки, чтобы определить радиус.
3. Нажмите, чтобы сформировать окружность.
4.  — отменить операцию.

Вспомогательные клавиши:

 — блокировать окружность в текущей ориентации (перед первым нажатием).

Инструмент «Дуга»

Используется для рисования дуг.

Управление инструментом:

1. Нажмите кнопку мыши в начальной точке дуги.
2. Переместите указатель.
3. Нажмите кнопку мыши в конечной точке дуги.
4. Переместите указатель перпендикулярно линии.
5. Нажмите, чтобы сформировать дугу.


6.  — отменить операцию.


Инструмент «Ластик»



Используется для удаления объектов.

Управление инструментом: нажмите объект, который необходимо удалить, или же нажмите и удерживайте кнопку мыши, перемещая указатель над объектами. Если отпустить кнопку мыши, все выделенные объекты будут удалены.

Вспомогательные клавиши:

 — скрыть объекты;

 — сгладить и смягчить края;

 +  — отменить сглаживание и смягчение объектов.

Инструмент «Рулетка»

Служит для многих целей: измерение расстояний, создание направляющих линий или точек, а также масштабирование модели.




Управление инструментом:

1. Нажмите кнопку мыши в начальной точке измерения.
2. Переместите указатель.
3. Нажмите кнопку мыши в конечной точке измерения.

Вспомогательные клавиши:

 — создание направляющих линий или направляющих точек;

 — блокировать текущее направление логического вывода рулетки;

 ,  ,  — блокировать определенное направление логического вывода линии (вверх — синяя, влево — зеленая, вправо — красная).


Инструмент «Заливка»


Осуществляет назначение материалов и цветов для объектов.

Управление инструментом:

1. Выберите библиотеку материалов в раскрывающемся списке браузера материалов.
2. Выберите материал в библиотеке материалов.
3. Нажмите грани, которые нужно окрасить.

Вспомогательные клавиши:

 — окрашивание всех граней, соединенных с используемой гранью, независимо от контекста;

 — окрашивание всех граней, соединенных с используемой гранью, в текущем контексте;


 +  — замена смежных объектов;

 — образец материала для окрашивания.

Инструмент «Тяни/Толкай»

Этот инструмент позволяет вдавливать и вытягивать грани объектов для добавления или уменьшения объема 3D-моделей.

Управление инструментом:

1. Нажмите грань.
2. Перемещайте указатель, чтобы увеличить (или уменьшить) объем.
3. Нажмите, чтобы завершить процесс.
4.  — отменить операцию.

Вспомогательные клавиши:

 — создание новой начальной грани;



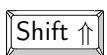
— выполнение операции «Тяни/Толкай» с растягиванием смежных граней.

Инструмент «Переместить»

Инструмент позволяет перемещать, растягивать или копировать объекты.
Управление инструментом:

1. Нажмите на объект. Также можно предварительно выбрать несколько объектов с помощью инструмента «Выбрать».
2. Переместите указатель в новое местоположение.
3. Нажмите кнопку мыши, чтобы завершить операцию.

Вспомогательные клавиши:



— блокировка перемещения к текущему направлению логического вывода;



— создание копии выбранного объекта;



— автоскладывание объектов;



,



,



— блокировка определенного направления логического вывода линии (вверх — синяя, влево — зеленая, вправо — красная).

Инструмент «Повернуть»


Осуществляет поворот, растяжение, искажение или копирование объектов вдоль закругленной линии.

Управление инструментом:

1. Нажмите на объект.
2. Перемещайте указатель по кругу, пока он не окажется в начальной точке поворота.
3. Чтобы разместить начальную точку поворота, нажмите кнопку мыши.

4. Перемещайте указатель по кругу, пока он не окажется в конечной точке поворота.
5. Нажмите кнопку мыши, чтобы завершить поворот.


Вспомогательные клавиши:

 — заблокировать инструмент в текущей ориентации (перед первым нажатием);

 — создать повернутую копию выбранного объекта.

Инструмент «Сдвиг»

Создает копии линий, которые равноотдалены от точек начала координат.
Управление инструментом:

1. Нажмите грань.
2. Переместите указатель.
3. Нажмите кнопку мыши, чтобы завершить операцию сдвига.
4.  — отменить операцию.

Инструмент «Орбита»


Осуществляет вращение камеры вокруг модели.

Управление инструментом:

1. Нажмите кнопку мыши в любой точке области рисования.
2. Перемещайте указатель в любом направлении, чтобы выполнить вращение вокруг центра области рисования.

Вспомогательные клавиши:

 — инструмент «Панорама»;

 — временно отключить силу притяжения (не удерживайте направление вертикальных краев вверх или вниз).

Инструмент «Панорама»

Осуществляет перемещение камеры (точки обзора) вертикально или горизонтально.

Управление инструментом:

1. Нажмите кнопку мыши в любой точке области рисования.
2. Перемещайте указатель в любом направлении для панорамирования.

Вспомогательные клавиши:



— включить предыдущий инструмент.

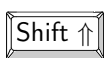
Инструмент «Масштаб»

Осуществляет приближение или удаление камеры (точки обзора).

Управление инструментом:

1. Нажмите и удерживайте кнопку мыши в любой точке области рисования.
2. Перетащите указатель вверх для увеличения (ближе к модели) или вниз для уменьшения (дальше от модели).

Вспомогательные клавиши



— изменить зону обзора;



— включить предыдущий инструмент.

Задание

По указанию преподавателя создать трехмерную геометрическую модель, состоящую из нескольких различных объектов, используя изученные теоретические сведения.

Контрольные вопросы

1. Что такое трехмерная графика?
2. Перечислите основные шаги, необходимые для получения трехмерного изображения на плоскости.
3. Перечислите инструменты, доступные в пакете Google SketchUp (не менее десяти).
4. Опишите использование инструментов для моделирования (не менее семи).

Литература

1. Google SketchUp [Электронный ресурс]. — Google SketchUp, 2010. — Режим доступа: <http://sketchup.google.com/index.html>. — Загл. с экрана.
2. Шикин, Е. В., Боресков, А. В. Компьютерная графика. Полигональные модели [Текст]. — М.: ДИАЛОГ-МИФИ, 2001. — 464 с.

4 Основы OpenGL

Цель работы

Целью выполнения данной работы является получение базовых знаний о спецификации OpenGL и приобретение основных навыков написания OpenGL-приложений с использованием библиотеки GLUT.

Теоретические сведения

Библиотека OpenGL

OpenGL (<http://www.opengl.org>) — это спецификация, описывающая набор графических операций и их результаты. Существует несколько реализаций этой спецификации в виде библиотек функций. Одна из распространенных реализаций OpenGL — библиотека Mesa (<http://www.mesa3d.org>).

С точки зрения программиста, OpenGL — это набор команд, которые описывают геометрические объекты и способ их отображения на экране. Типичная программа, использующая OpenGL начинается с определения окна, в котором будет происходить отображение. Затем создается контекст OpenGL и ассоциируется с этим окном. Далее программист может использовать функции OpenGL для рисования графических примитивов и задания режимов их отображения.

OpenGL обрабатывает и выводит графические примитивы (точки, отрезки, многоугольники и др.) с учетом некоторого числа выбранных режимов отображения (каждый режим может быть изменен независимо от других). Определение примитивов, выбор режимов и другие операции осуществляется с помощью вызовов соответствующих функций.

Примитивы определяются набором из одной или более вершин (vertex). Вершина определяет точку, конец грани, угол многоугольника. С каждой вершиной ассоциируются некоторые данные (координаты, цвет, нормаль, текстурные координаты). В подавляющем большинстве случаев каждая вершина обрабатывается независимо от других.

Команды OpenGL всегда обрабатываются в том порядке, в котором они поступают, хотя могут происходить задержки перед тем, как проявится эффект от их выполнения.

Для обеспечения интуитивно понятных названий в OpenGL полное имя функции имеет вид:

```
type glCommandName[1 2 3 4][b s i f d ub us ui][v]
    (type1 arg1, ..., typeN argN)
```

Таким образом, имя состоит из нескольких частей.

gl — это имя библиотеки, в которой описана эта функция: для базовых функций OpenGL, функций из библиотек GLU, GLUT, GLAUX это **gl**, **glu**, **glut**, **aux** соответственно.

CommandName — имя команды.

[1 2 3 4] — число аргументов команды.

[b s i f d ub us ui] — тип аргумента: символ **b** означает тип **GLbyte** (аналог **char** в C/C++), символ **f** — тип **GLfloat** (аналог **float**), символ **i** — тип **GLint** (аналог **int**) и так далее. Полный список типов и их описание можно посмотреть в файле **gl.h**.

[v] — наличие этого символа показывает, что в качестве параметров функции используется указатель на массив значений.

Символы в квадратных скобках в некоторых названиях не используются. Например, команда **glVertex2i()** описана как базовая в библиотеке OpenGL, и использует в качестве параметров два целых числа, а команда **glColor3fv()** использует в качестве параметра указатель на массив из трех вещественных чисел.

Библиотека GLUT

Рассмотрим построение приложения при помощи библиотеки GLUT (<http://www.opengl.org/resources/libraries/glut>) (GL Utility Toolkit), получив-

шей в последнее время широкое распространение. Эта библиотека обеспечивает единый интерфейс для работы с окнами вне зависимости от платформы, поэтому описываемая ниже структура приложения остается неизменной для операционных систем Windows, Linux и многих других.

Инициализация проводится с помощью функции

```
glutInit(int *argc, char **argv).
```

Переменная `argc` есть указатель на стандартную переменную `argc` описываемую в функции `main()`, а `argv` — указатель на параметры, передаваемые программе при запуске, который описывается там же. Эта функция проводит необходимые начальные действия для построения окна приложения, и только несколько функций GLUT могут быть вызваны до нее. К ним относятся:

```
glutInitWindowPosition(int x, int y),  
glutInitWindowSize(int width, int height),  
glutInitDisplayMode(unsigned int mode).
```

Первые две функции задают соответственно положение и размер окна, а последняя функция определяет различные режимы отображения информации, которые могут совместно задаваться с использованием операции побитового «или» (`<|>`):

`GLUT_RGBA` — режим RGBA. Используется по умолчанию, если не указаны явно режимы `GLUT_RGBA` или `GLUT_INDEX`.

`GLUT_RGB` — то же, что и `GLUT_RGBA`.

`GLUT_INDEX` — режим индексированных цветов (использование палитры). Отменяет `GLUT_RGBA`.

`GLUT_SINGLE` — окно с одиночным буфером. Используется по умолчанию.

`GLUT_DOUBLE` — окно с двойным буфером. Отменяет `GLUT_SINGLE`.

`GLUT_DEPTH` — окно с буфером глубины.

Это неполный список параметров для данной функции, однако для большинства случаев этого бывает достаточно.

Двойной буфер обычно используют для анимации, сначала рисуя что-нибудь в одном буфере, а затем меняя их местами, что позволяет избежать

мерцания. Буфер глубины или z-буфер используется для удаления невидимых линий и поверхностей.

Функции библиотеки GLUT реализуют так называемый событийно-управляемый механизм. Это означает, что есть некоторый внутренний цикл, который запускается после соответствующей инициализации и обрабатывает, один за другим, все события, объявленные во время инициализации. К событиям относятся: щелчок мыши, закрытие окна, изменение свойств окна, передвижение курсора, нажатие клавиши, и «пустое» (idle) событие, когда ничего не происходит. Для проведения периодической проверки совершения того или иного события надо зарегистрировать функцию, которая будет его обрабатывать. Для этого используются функции вида:

```
void glutDisplayFunc(void (*func)(void)),  
void glutReshapeFunc(void (*func)(int width, int height)),  
void glutKeyboardFunc(void (*func)(unsigned char key,  
int x, int y)),  
void glutMouseFunc(void (*func)(int button, int state,  
int x, int y)),  
void glutIdleFunc(void (*func)(void)).
```

То есть параметром для них является имя соответствующей функции заданного типа. С помощью `glutDisplayFunc()` задается функция рисования для окна приложения, которая вызывается при необходимости создания или восстановления изображения. Для явного указания, что окно надо обновить, иногда удобно использовать функцию `void glutPostRedisplay(void)`.

Через `glutReshapeFunc()` устанавливается функция обработки изменения размеров окна пользователем, которой передаются новые размеры.

`void glutKeyboardFunc()` определяет обработчика событий от клавиатуры. Примеры событий: `GLUT_KEY_F1` — пользователь нажал клавишу «F1» на клавиатуре, `GLUT_KEY_LEFT` — пользователь нажал клавишу «стрелка влево» на клавиатуре. Названия остальных событий можно посмотреть в заголовочном файле `glut.h`.

`glutMouseFunc()` определяет обработчика событий от мыши. Пример со-

бытия: GLUT_LEFT_BUTTON — пользователь нажал левую кнопку мыши. Названия остальных событий можно посмотреть в заголовочном файле `glut.h`.

`glutIdleFunc()` задает функцию, которая будет вызываться каждый раз, когда нет событий от пользователя.

Контроль всех событий происходит внутри бесконечного цикла в функции `void glutMainLoop(void)` которая обычно вызывается в конце любой программы, использующей GLUT. Структура приложения, использующего анимацию, будет следующей:

```
#include <GL/glut.h>

void MyIdle(void)
{
    /* Код, меняющий переменные, определяющие следующий кадр */
}

void MyDisplay(void)
{
    /* Код OpenGL, который отображает кадр */

    /* После рисования переставляем буфера */
    glutSwapBuffers();
}

int main(int argc, char **argv)
{
    /* Инициализация GLUT */
    glutInit(&argc, argv);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(0, 0);

    /* Открытие окна */
}
```

```

glutCreateWindow("My OpenGL Application");

/* Выбор режима: двойной буфер и RGBA цвета */
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);

/* Регистрация вызываемых функций */
glutDisplayFunc(MyDisplay);
glutIdleFunc(MyIdle);

/* Запуск механизма обработки событий */
glutMainLoop();
}

```

В случае если приложение должно строить статичное изображение, можно заменить `GLUT_DOUBLE` на `GLUT_SINGLE`, так как одного буфера в этом случае будет достаточно, и убрать вызов функции `glutIdleFunc()`.

Для успешной сборки приложения необходимо указать, что используются функции из той или иной реализации OpenGL. Это можно сделать в среде разработки (настройки проекта) или с помощью ключей компилятора — необходимо указать ключи и для реализации OpenGL, и для дополнительных библиотек (в данном случае, GLUT). В случае использования компилятора gcc необходимо добавить следующие ключи: `-lopengl32 -lglu32 -lglut32`.

Вершины и примитивы

Положение вершины определяются заданием их координат в двух-, трех-, или четырехмерном пространстве (однородные координаты). Это реализуется с помощью нескольких версий функции `glVertex`:

```

void glVertex[2 3 4][s i f d](type coords)
void glVertex[2 3 4][s i f d]v(type *coords)

```

Каждая функция задает 4 координаты вершины: `x`, `y`, `z` и `w`. Функция `glVertex2` получает значения `x` и `y`. Координата `z` в таком случае устанавливается по умолчанию равной 0, а координата `w` равной 1. `Vertex3` получает

координаты **x**, **y**, **z** и заносит в координату **w** значение 1. **Vertex4** позволяет задать все 4 координаты.

Для ассоциации с вершинами цветов, нормалей, и текстурных координат используются текущие значения соответствующих данных. Эти значения могут быть изменены в любой момент путем использования соответствующих функций.

Для задания текущего цвета вершины используются функции

```
void glColor[3 4][b s i f](GLtype components)
```

```
void glColor[3 4][b s i f]v(GLtype components)
```

Первые три параметра задают красную, зеленую и синюю компоненты цвета, а последний параметр определяет α -компоненту, которая задает уровень прозрачности объекта. Если в названии функции указан тип «f» (**float**), то значения всех параметров должны принадлежать отрезку $[0, 1]$, при этом по умолчанию значение α -компоненты устанавливается равным 1.0, что соответствует полной непрозрачности. Если указан тип «ub» (**unsigned byte**), то значения должны лежать в отрезке $[0, 255]$.

Разным вершинам можно назначать различные цвета и тогда будет проводиться линейная интерполяция цветов по поверхности примитива.

Для управления режимом интерполяции цветов используется функция

```
void glShadeModel(GLenum mode)
```

вызов которой с параметром **GL_SMOOTH** включает интерполяцию (установка по умолчанию), а с **GL_FLAT** — отключает.

Аналогичным образом можно определить нормаль в вершине, используя функции

```
void glNormal3[b s i f d](type coords)
```

```
void glNormal3[b s i f d]v(type coords)
```

Задаваемый вектор может не иметь единичной длины, но он будет нормироваться автоматически в режиме нормализации, который включается вызовом команды **glEnable(GL_NORMALIZE)**. Функции

```
void glEnable(GLenum mode) void glDisable(GLenum mode)
```

производят включение и отключение того или иного режима работы конвей-

ера OpenGL.

Мы рассмотрели задание атрибутов одной вершины. Однако чтобы задать какую-нибудь фигуру, одних координат вершин недостаточно, и эти вершины надо объединить в одно целое, определив необходимые свойства. Для этого в OpenGL используется понятие примитивов, к которым относятся точки, линии, связанные или замкнутые линии, треугольники и так далее. Задание примитива происходит внутри так называемых командных скобок:

```
void glBegin(GLenum mode);
```

```
void glEnd(void);
```

Параметр `mode` определяет тип примитива (см. рис. 4), который задается внутри и может принимать следующие значения:

`GL_POINTS` — каждая вершина задает координаты некоторой точки.

`GL_LINES` — каждая отдельная пара вершин определяет отрезок; если задано нечетное число вершин, то последняя вершина игнорируется.

`GL_LINE_STRIP` — каждая следующая вершина задает отрезок вместе с предыдущей.

`GL_LINE_LOOP` — отличие от предыдущего примитива только в том, что последний отрезок определяется по-следней и первой вершиной, образуя замкнутую ломаную.

`GL_TRIANGLES` — каждая отдельная тройка вершин определяет треугольник; если задано не кратное трем число вершин, то последние вершины игнорируются.

`GL_TRIANGLE_STRIP` — каждая следующая вершина задает треугольник вместе с двумя предыдущими.

`GL_TRIANGLE_FAN` — треугольники задаются первой и каждой следующей парой вершин (пары не пересекаются).

`GL_QUADS` — каждая отдельная четверка вершин определяет четырехугольник; если задано не кратное четырем число вершин, то последние вершины игнорируются.

GL_QUAD_STRIP — четырехугольник с номером n определяется вершинами с номерами $2n - 1, 2n, 2n + 2, 2n + 1$.

GL_POLYGON — последовательно задаются вершины выпуклого многоугольника.

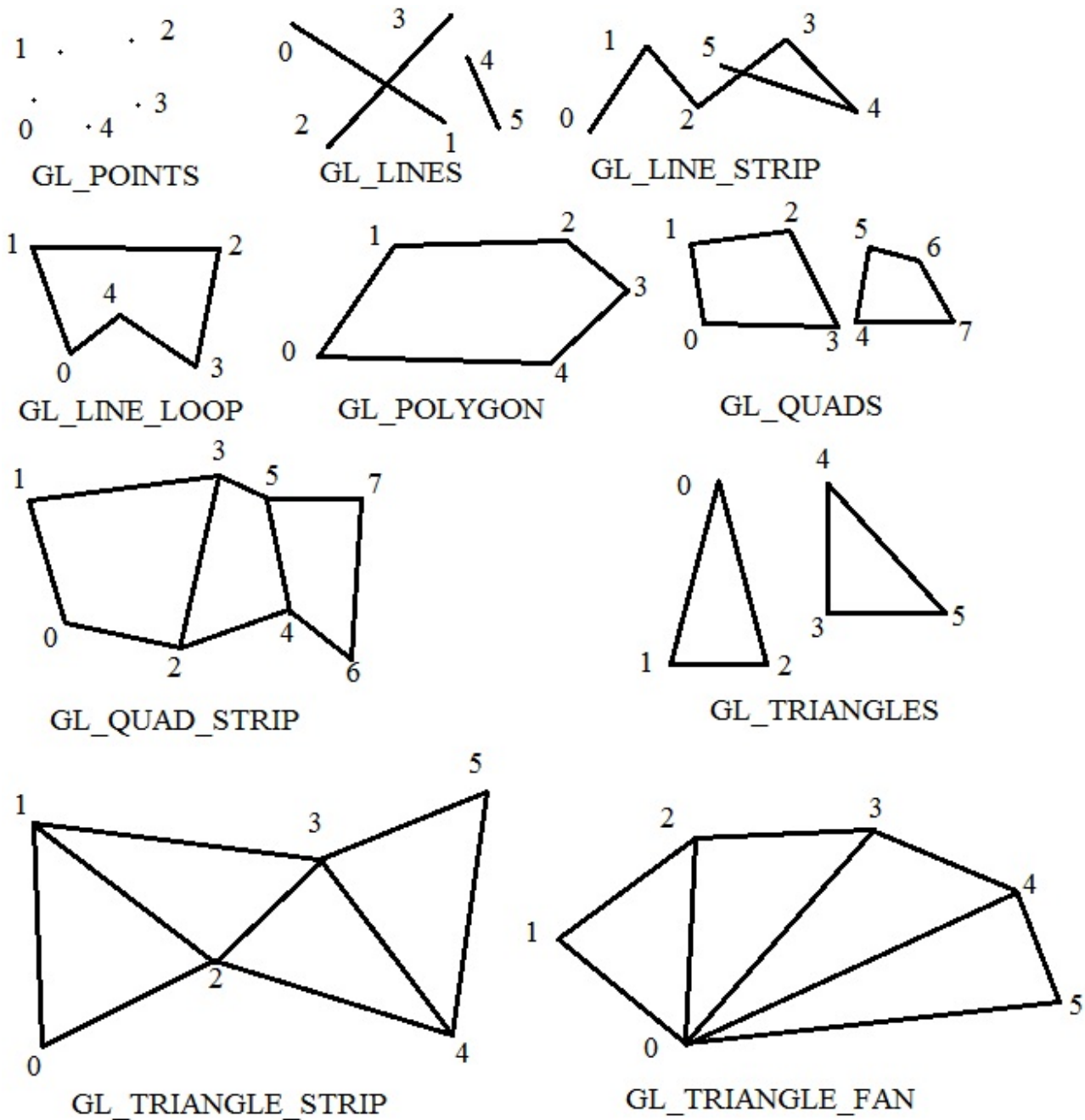


Рис. 4: Примитивы OpenGL.

Например, чтобы нарисовать треугольник с разными цветами в вершинах,

достаточно написать:

```
GLfloat BlueCol[3] = {0,0,1};

glBegin(GL_TRIANGLE);
glColor3f(1.0, 0.0, 0.0);    /* красный */
glVertex3f(0.0, 0.0, 0.0);
glColor3ub(0, 255, 0);       /* зеленый */
glVertex3f(1.0, 0.0, 0.0);
glColor3fv(BlueCol);         /* синий */
glVertex3f(1.0, 1.0, 0.0);
glEnd();
```

Кроме задания самих примитивов можно определить метод их отображения на экране (под примитивами в данном случае понимаются многоугольники). Однако сначала надо определить понятие лицевых и обратных граней. Под гранью понимается одна из сторон многоугольника, и по умолчанию лицевой считается та сторона, вершины которой обходятся против часовой стрелки. Направление обхода вершин лицевых сторон можно изменить вызовом функции

```
void glFrontFace(GLenum mode)
```

со значением параметра `mode` равным `GL_CW`, а отменить — с `GL_CCW`.

Чтобы изменить метод отображения многоугольника используется функция

```
void glPolygonMode (GLenum face, GLenum mode)
```

Параметр `mode` определяет, как будут отображаться многоугольники, а параметр `face` устанавливает тип многоугольников и может принимать следующие значения:

`GL_FRONT` — для лицевых граней

`GL_BACK` — для обратных граней

`GL_FRONT_AND_BACK` — для всех граней

Параметр `mode` может быть равен:

`GL_POINT` — при таком режиме будут отображаться только вершины многоугольников.

`GL_LINE` — при таком режиме многоугольник будет представляться набором отрезков.

`GL_FILL` — при таком режиме многоугольники будут закрашиваться текущим цветом с учетом освещения, и этот режим установлен по умолчанию.

Кроме того, можно указывать, какой тип граней отображать на экране. Для этого сначала надо установить соответствующий режим вызовом функции `glEnable(GL_CULL_FACE)`, а затем выбрать тип отображаемых граней с помощью функции `void glCullFace(GLenum mode)`. Вызов с параметром `GL_FRONT` приводит к удалению из изображения всех лицевых граней, а с параметром `GL_BACK` — обратных (установка по умолчанию).

Кроме рассмотренных стандартных примитивов в библиотеках `GLU` и `GLUT` описаны более сложные фигуры, такие как сфера, цилиндр, диск (в `GLU`) и сфера, куб, конус, тор, тетраэдр, додекаэдр, икосаэдр, октаэдр и чайник (в `GLUT`). Автоматическое наложение текстуры предусмотрено только для фигур из библиотеки `GLU`.

Например, чтобы нарисовать сферу или цилиндр, надо сначала создать объект специального типа `GLUQuadricObj` с помощью функции

```
GLUQuadricObj* gluNewQuadric(void);
```

а затем вызвать соответствующую функцию:

```
void gluSphere(GLUQuadricObj * qobj, GLdouble radius,  
GLint slices, GLint stacks),
```

```
void gluCylinder(GLUQuadricObj * qobj, GLdouble baseRadius,  
GLdouble topRadius, GLdouble height, GLint slices, GLint stacks),
```

где параметр `slices` задает число разбиений вокруг оси z , а `stacks` — вдоль оси z .

Преобразования координат

В OpenGL используются как основные три системы координат: левосторонняя, правосторонняя и оконная. Первые две системы являются трехмерными и отличаются друг от друга направлением оси z : в правосторонней она направлена на наблюдателя, а в левосторонней — вглубь экрана. Расположение осей x и y аналогично описанному выше. Левосторонняя система используется для задания значений параметрам команды `gluPerspective()`, `glOrtho()`, которые будут рассмотрены ниже, а правосторонняя или мировая система координат во всех остальных случаях. Отображение трехмерной информации происходит в двумерную оконную систему координат.

Для задания различных преобразований объектов сцены в OpenGL используются операции над матрицами, при этом различают три типа матриц: видовая, проекций и текстуры. Все они имеют размер 4×4 . Видовая матрица определяет преобразования объекта в мировых координатах, такие как параллельный перенос, изменение масштаба и поворот. Матрица проекций задает как будут проецироваться трехмерные объекты на плоскость экрана (в оконные координаты), а матрица текстуры определяет наложение текстуры на объект.

Для того чтобы выбрать, какую матрицу надо изменить, используется функция `void glMatrixMode(GLenum mode)`, вызов которой со значением параметра `mode` равным `GL_MODELVIEW`, `GL_PROJECTION`, `GL_TEXTURE` включает режим работы с видовой, проекций и матрицей текстуры соответственно. Для вызова функций, задающих матрицы того или иного типа необходимо сначала установить соответствующий режим.

Для определения элементов матрицы текущего типа вызывается функция

```
void glLoadMatrix[f d] (GLtype *m)
```

где `m` указывает на массив из 16 элементов типа `float` или `double` в соответствии с названием команды, при этом сначала в нем должен быть записан первый столбец матрицы, затем второй, третий и четвертый.

Функция `void glLoadIdentity(void)` заменяет текущую матрицу на единичную. Часто нужно сохранить содержимое текущей матрицы для дальней-

шего использования, для чего используют функции

```
void glPushMatrix(void)
```

```
void glPopMatrix(void)
```

Они записывают и восстанавливают текущую матрицу из стека, причем для каждого типа матриц стек свой. Для видовых матриц его глубина равна как минимум 32, а для двух оставшихся типов — как минимум 2.

Для умножения текущей матрицы слева на другую матрицу используется функция

```
void glMultMatrix[f d] (GLtype *m)
```

где *m* должен задавать матрицу размером 4×4 в виде массива с описанным расположением данных. Однако обычно для изменения матрицы того или иного типа удобно использовать специальные команды, которые по значениям своих параметров создают нужную матрицу и перемножают ее с текущей. Чтобы сделать текущей созданную матрицу, надо перед вызовом этой функции вызвать `glLoadIdentity()`.

В целом, для отображения трехмерных объектов сцены в окно приложения используется показанная на рис. 5 последовательность действий.

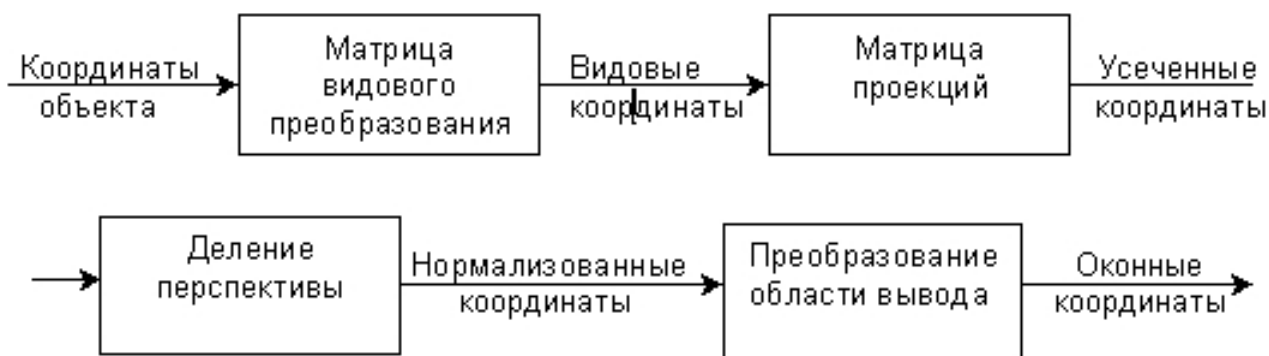


Рис. 5: Конвейер OpenGL.

К видовым преобразованиям будем относить перенос, поворот и изменение масштаба вдоль координатных осей. Для проведения этих операций достаточно умножить на соответствующую матрицу каждую вершину объекта и получить измененные координаты этой вершины:

$$(x', y', z', 1)^T = M(x, y, z, 1)^T$$

где M — матрица видового преобразования. Перспективное преобразование и проектирование производится аналогично. Сама матрица может быть создана с помощью следующих функций:

```
void glTranslate[f d](GLtype x, GLtype y, GLtype z),
void glRotate[f d](GLtype angle, GLtype x, GLtype y, GLtype z),
void glScale[f d](GLtype x, GLtype y, GLtype z).
```

`glTranlsate..()` производит перенос объекта, прибавляя к координатам его вершин значения своих параметров.

`glRotate..()` производит поворот объекта против часовой стрелки на угол `angle` (измеряется в градусах) вокруг вектора (x, y, z) .

`glScale..()` производит масштабирование объекта (сжатие или растяжение), домножая соответствующие координаты его вершин на значения своих параметров.

Все эти преобразования будут применяться к примитивам, описания которых будут находиться ниже в программе. В случае если надо, например, повернуть один объект сцены, а другой оставить неподвижным, удобно сначала сохранить текущую видовую матрицу в стеке командой `glPushMatrix()`, затем вызвать `glRotate..()` с нужными параметрами, описать примитивы, из которых состоит этот объект, а затем восстановить текущую матрицу командой `glPopMatrix()`.

Кроме изменения положения самого объекта иногда бывает нужно изменить положение точки наблюдения, что однако также приводит к изменению видовой матрицы. Это можно сделать с помощью функции

```
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
GLdouble centerx, GLdouble centery, GLdouble centerz,
GLdouble upx, GLdouble upy, GLdouble upz),
```

где точка $(eyex, eyey, eyez)$ определяет точку наблюдения, $(centerx, centery, centerz)$ задает центр сцены, который будет проектироваться в центр области вывода, а вектор (upx, upy, upz) задает положительное на-

правление оси z , определяя поворот камеры. Если, например, камеру не надо поворачивать, то задается значение $(0, 1, 0)$, а со значением $(0, -1, 0)$ сцена будет перевернута.

Фактически, эта функция совершает перенос и поворот объектов сцены, но в таком виде задавать параметры бывает удобнее.

В OpenGL существуют ортографическая (параллельная) и перспективная проекция. Первый тип проекции может быть задан вызовом функций

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
GLdouble top, GLdouble near, GLdouble far),
```

```
void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom,
GLdouble top).
```

Первая функция создает матрицу проекции в усеченный объем видимости (параллелограмм видимости) в левосторонней системе координат. Параметры команды задают точки $(left, bottom, -near)$ и $(right, top, -near)$, которые отвечают левому нижнему и правому верхнему углам окна вывода. Параметры **near** и **far** задают расстояние до ближней и дальней плоскостей отсечения по дальности от точки $(0, 0, 0)$ и могут быть отрицательными.

Во второй функции, в отличие от первой, значения **near** и **far** устанавливаются равными -1 и 1 соответственно.

Перспективная проекция определяется вызовом функции

```
void gluPerspective(GLdouble angley, GLdouble aspect,
GLdouble znear, GLdouble zfar),
```

которая задает усеченный конус видимости в левосторонней системе координат. Параметр **angley** определяет угол видимости в градусах по оси y и должен находиться в диапазоне от 0 до 180. Угол видимости вдоль оси x задается параметром **aspect**, который обычно задается как отношение сторон области вывода. Параметры **zfar** и **znear** задают расстояние от наблюдателя до плоскостей отсечения по глубине и должны быть положительными. Чем больше отношение $zfar/znear$, тем хуже в буфере глубины будут различаться расположенные рядом поверхности, так как по умолчанию в него будет записываться «сжатая» глубина в диапазоне от 0 до 1.

После применения матрицы проекций на вход следующего преобразования подаются так называемые усеченные (clip) координаты, для которых значения всех компонент $(x_c, y_c, z_c, w_c)^T$ находятся в отрезке $[-1, 1]$. После этого находятся нормализованные координаты вершин по формуле:

$$(x_n, y_n, z_n)^T = (x_c/w_c, y_c/w_c, z_c/w_c)^T.$$

Область вывода представляет из себя прямоугольник в оконной системе координат, размеры которого задаются параметрами следующей функции:

```
void glViewport(GLint x, GLint y, GLint width, GLint height).
```

Значения всех параметров задаются в пикселах и определяют ширину и высоту области вывода с координатами левого нижнего угла (x, y) в оконной системе координат. Размеры оконной системы координат определяются текущими размерами окна приложения, точка $(0, 0)$ находится в левом нижнем углу окна. Используя параметры функции `glViewport()`, вычисляются оконные координаты центра области вывода (o_x, o_y) по формулам

$$o_x = x + width/2, \quad o_y = y + height/2.$$

Пусть $p_x = width$, $p_y = height$, тогда можно найти оконные координаты каждой вершины:

$$(x_w, y_w, z_w)^T = \left(\frac{p_x}{2}x_n + o_x, \frac{p_y}{2}y_n + o_y, \frac{f-n}{2}z_n + \frac{n+f}{2} \right)^T$$

При этом целые положительные величины n и f задают минимальную и максимальную глубину точки в окне и по умолчанию равны 0 и 1 соответственно. Глубина каждой точки записывается в специальный буфер глубины (z-буфер), который используется для удаления невидимых линий и поверхностей. Установить значения n и f можно вызовом функции

```
void glDepthRange(GLclampd n, GLclampd f).
```

Функция `glViewport()` обычно используется в функции, зарегистрированной с помощью команды `glutReshapeFunc()`, которая вызывается, если пользователь изменяет размеры окна приложения, изменяя соответствующим об-

разом область вывода.

Задание

Написать и отладить программу для демонстрации структуры GLUT-приложения и простейших основ OpenGL. Результатом работы программы должна быть трехмерная сцена, содержащая набор цветных примитивов разных типов. Предусмотреть обработку событий от мыши и клавиатуры (в частности, обеспечить навигацию по сцене с помощью клавиш со стрелками на клавиатуре).

Контрольные вопросы

1. Перечислить этапы обработки данных используемые в конвейере OpenGL.
2. Пояснить смысл каждой части OpenGL-команды.
3. Перечислить и охарактеризовать функции GLUT (не менее семи).
4. Описать, как примитивы задаются в тексте программы.
5. Перечислить типы примитивов (не менее восьми) и охарактеризовать их.
6. Перечислить методы отображения примитивов (не менее шести).
7. Перечислить и охарактеризовать функции для работы с матрицами преобразований (не менее шести).
8. Перечислить и охарактеризовать функции для работы с проекциями и областью вывода.

Литература

1. Шикин, Е. В., Боресков, А. В. Компьютерная графика. Полигональные модели [Текст]. — М.: ДИАЛОГ-МИФИ, 2001. — 464 с.

2. Эйнджел, Э. Интерактивная компьютерная графика. Вводный курс на базе OpenGL [Текст]. — М.: Издательский дом «Вильямс», 2001. — 592 с.
3. Хилл, Ф. Программирование компьютерной графики. Для профессионалов [Текст]. — СПб.: Питер, 2002. — 1088 с.

5 Формат хранения графической информации BMP

Цель работы

Целью выполнения данной работы является получение базовых знаний о хранении растровой графической информации на примере формата BMP.

Теоретические сведения

BMP (от англ. bitmap) — формат хранения растровых изображений. Файлы формата BMP могут иметь расширения .bmp, .dib и .rle.

Структурно BMP-файл состоит из четырёх частей:

1. Заголовок файла (BITMAPFILEHEADER).
2. Заголовок изображения (BITMAPINFOHEADER, может отсутствовать).
3. Палитра (может отсутствовать).
4. Само изображение.

Заголовок файла

Заголовок файла можно представить в виде структуры BITMAPFILEHEADER, содержащей информацию о типе, размере и представлении данных в файле.

```
typedef struct tagBITMAPFILEHEADER {  
    WORD    bfType;  
    DWORD   bfSize;  
    WORD    bfReserved1;  
    WORD    bfReserved2;  
    DWORD   bfOffBits;  
} BITMAPFILEHEADER;
```

Поля структуры имеют следующий смысл:

bfType — тип файла: 0x4D42 (символы «BM»).

bfSize — размер файла в байтах. Обычно содержимое этого поля игнорируется, так как из-за ошибки в документации старые приложения устанавливали в этом поле неправильное значение.

bfReserved1 — зарезервирован, содержит ноль.

bfReserved2 — зарезервирован, содержит ноль.

bfOffBits — содержит смещение в байтах от начала структуры **BITMAPFILEHEADER** до непосредственно битов изображения. **Область изображения не обязательно должна быть расположена сразу вслед за заголовками файла или таблицей цветов (если она есть)!**

Тип **WORD** должен иметь размер 16 бит, типы **DWORD** и **LONG** — 32 бита, все типы данных целые, порядок байтов подразумевается **little endian** (от младшего к старшему).

Заголовок изображения

Опишем наиболее простой вариант заголовка изображения в виде структуры **BITMAPINFOHEADER**.

```
typedef struct tagBITMAPINFOHEADER {  
    DWORD   biSize;  
    LONG    biWidth;  
    LONG    biHeight;  
    WORD    biPlanes;  
    WORD    biBitCount;  
    DWORD   biCompression;  
    DWORD   biSizeImage;  
    LONG    biXPelsPerMeter;  
    LONG    biYPelsPerMeter;
```

```
DWORD  biClrUsed;  
DWORD  biClrImportant;  
} BITMAPINFOHEADER;
```

biSize — размер структуры в байтах.

biWidth — ширина изображения в пикселях.

biHeight — высота изображения в пикселях. Если содержит положительное значение — изображение записано в порядке снизу вверх, нулевой пиксель в нижнем левом углу. Если значение отрицательное — изображение записано сверху вниз, нулевой пиксель в верхнем левом углу изображения.

biPlanes — количество плоскостей в битовом изображении. Содержимое этого поля должно быть равно единице.

biBitCount — указывает количество бит на пиксель. Может принимать следующие значения:

- 1 — изображение монохромное. Член **bmiColors** структуры **BITMAPINFO** содержит два элемента. Каждый бит изображения представляет один пиксель; если бит равен нулю — пиксель имеет цвет первого элемента таблицы **bmiColors**, иначе — цвет второго.
- 4 — шестнадцатичетное изображение. Пиксели определяются четырёхбитными индексами, каждый байт изображения содержит информацию о двух пикселях — старшие 4 бита для первого, оставшиеся — для второго.
- 8 — в палитре содержится до 256 цветов, каждый байт изображения хранит индекс в палитре для одного пикселя.
- 16 — если поле **biCompression** содержит значение **BI_RGB**, файл не содержит палитры. Каждые два байта изображения хранят интенсивность красной, зелёной и синей компоненты одного пикселя. При этом старший бит не используется, на каждую компоненту отведено 5 бит: **ORRRRRGGGGGBBBBB**. Если поле **biCompression** со-

держит значение `BI_BITFIELDS`, палитра хранит три двухбайтных значения, определяющих маску для каждой из трёх компонент цвета. Каждый пиксель изображения представлен двухбайтным значением, из которого с помощью масок извлекаются цветовые компоненты.

24 — палитра не используется, каждая тройка байт изображения представляет один пиксель, по байту для интенсивности синего, зелёного и красного канала соответственно.

32 — Если поле `biCompression` содержит значение `BI_RGB`, изображение не содержит палитры. Каждые четыре байта изображения представляют один пиксель, по байту для интенсивности синего, зелёного и красного канала соответственно. Старший байт каждой четвёрки не используется. Если поле `biCompression` содержит значение `BI_BITFIELDS`, в палитре хранятся три четырёхбайтных цветовых маски — для красной, зелёной и синей компоненты. Каждый пиксель изображения представлен четырьмя байтами.

`biCompression` — указывает тип сжатия для сжатых изображений.

`biSizeImage` — указывает размер изображения в байтах. Может содержать ноль для `BI_RGB`-изображений.

`biXPelsPerMeter` — указывает горизонтальное разрешение в пикселях на метр для целевого устройства. Приложение может использовать это значение для выбора из группы ресурсов изображения, наиболее подходящего для текущего устройства.

`biYPelsPerMeter` — указывает вертикальное разрешение в пикселях на метр для целевого устройства.

`biClrUsed` — указывает количество используемых цветовых индексов в палитре. Если значение равно нулю — изображение использует максимально доступное количество индексов, в соответствии со значением `biBitCount` и методом сжатия, указанным в `biCompression`. Если содержит ненулевое значение и `biBitCount` меньше 16, `biClrUsed` ука-

зывает количество цветов, к которым будет обращаться драйвер устройства или приложение. Если `biBitCount` больше или равен 16, `biClrUsed` указывает размер палитры, используемой для оптимизации работы системных палитр. Если `biBitCount` равен 16 или 32, оптимальная палитра следует сразу после трёх четырёхбайтных маск. В упакованном изображении массив пикселей следует сразу после структуры `BITMAPINFO`, `biClrUsed` должен содержать ноль, либо реальный размер палитры.

`biClrImportant` — указывает количество индексов, необходимых для отображения изображения. Если содержит ноль — все индексы одинаково важны.

Палитра

Палитра может содержать последовательность четырёхбайтовых полей по числу доступных цветов (256 для 8-битного изображения). Три младшие байта каждого поля определяют интенсивность красной, зелёной и синей компоненты цвета, старший байт не используется. Каждый пиксель изображения описан в таком случае одним байтом, содержащим номер поля палитры, в котором сохранен цвет этого пикселя.

Если пиксель изображения описывается 16-битным числом, палитра может хранить три двухбайтных значения, каждое из которых пределяет маску для извлечения из 16-битного пикселя красной, зелёной и синей компонент цвета.

Файл BMP может не содержать палитры, если в нём хранится несжатое полноцветное изображение.

Данные изображения

Данные изображения хранятся в виде последовательности пикселей, записанных в том или ином виде. Пиксели хранятся построчно, снизу вверх. **Каждая строка изображения дополняется нулями до длины, кратной четырём байтам!**

В bmp-файлах с глубиной цвета 24 бита, байты цвета каждого пикселя хранятся в порядке BGR (Blue, Green, Red).

В bmp-файлах с глубиной цвета 32 бита, байты цвета каждого пиксела хранятся в порядке BGRA (Blue, Green, Red, Alpha).

Битность изображения. В зависимости от количества представляемых цветов, на каждую точку отводится от 1 до 48 битов:

1 бит — монохромное изображение (два цвета).

2 бита — 4 возможных цвета (режимы работы CGA).

4 бита — 16-цветное изображение (режимы работы EGA).

8 бит (1 байт) — 256 цветов, последний из режимов, поддерживавших индексированные цвета (см. ниже).

16 бит (2 байта) — режим HiColor, 65536 возможных оттенков.

24 бита (3 байта) — TrueColor. В связи с тем, что 3 байта не очень хорошо соотносятся с степенями двойки (особенно при хранении данных в памяти, где выравнивание данных по границе слова имеет значение), вместо него часто используют 32-битное изображение. В режиме TrueColor на каждый из трёх каналов (в режиме RGB) отводится по 1 байту (256 возможных значений), общее количество цветов равно 16777216.

32 бита (4 байта) — этот режим практически аналогичен TrueColor, четвёртый байт обычно не используется, или в нём располагается альфа-канал (прозрачность).

48 бит (6 байт) — редко используемый формат с повышенной точностью передачи цвета, поддерживается относительно малым количеством программ и оборудования.

Индексированные цвета. При количестве бит от 1 до 8 на каждый пиксел может использоваться специальный режим индексированных цветов. В этом случае число, соответствующее каждому пикселу указывает не на цвет, а на номер цвета в палитре. Благодаря использованию палитры имеется возможность адаптировать изображение к цветам, присутствующим на изобра-

жении. В таком случае изображение ограничено не заданными цветами, а максимальным количеством одновременно используемых цветов.

Задание

Написать программу для чтения и просмотра BMP-файлов. Предусмотреть возможность работы с файлами, использующими индексированные цвета.

Контрольные вопросы

1. Перечислить и охарактеризовать все части BMP-файла.
2. Записать структуры для заголовка файла и заголовка изображения. Пояснить смысл всех входящих в них полей.
3. Объяснить, как хранятся палитра и данные изображения в BMP-файле.
4. Объяснить, что такое индексированные цвета, и как они могут использоваться.

Литература

1. Шикин, Е. В., Боресков, А. В. Компьютерная графика. Полигональные модели [Текст]. — М.: ДИАЛОГ-МИФИ, 2001. — 464 с.
2. Роджерс, Д. Алгоритмические основы машинной графики [Текст]. — М.: Мир, 1989. — 512 с.

6 Обработка растровых изображений

Цель работы

Целью выполнения данной работы является получение базовых знаний о методах обработки растровых изображений и приобретение навыков по их реализации.

Теоретические сведения

Основным типом изображения при обработке, как правило, выступают полутоновые изображения. Процесс трансформации изображения из одного типа в другой может быть осуществлен по следующей схеме: полноцветные \rightarrow полутоновые \rightarrow бинарные.

Получить полутоновое изображение из полноцветного можно с помощью формулы

$$I = 0.3R + 0.59G + 0.11B, \quad (1)$$

где R , G , B — значения красного, зеленого и синего цветов соответственно, I — значение интенсивности оттенков какого-либо цвета, например, серого. Значение 0 соответствует черному цвету (отсутствие интенсивности), значение 255 — белому (максимальная интенсивность).

Для цифровой обработки изображений применяют методы разных классов. Мы рассмотрим наиболее простой класс: точечные процессы. Точечные процессы — алгоритмы, которые изменяют значение пикселя только в зависимости от его собственного значения, а иногда и от положения. Никакие другие значения пикселя не включаются в преобразование. Индивидуальные элементы изображения заменяются новыми значениями, которые алгоритмически связаны с исходным значением элемента изображения. Как результат алгоритмических отношений между исходным и новым значением элемента изображения, точечные процессы могут в общем случае выполняться и в обратном порядке. Точечные процессы — фундаментальные операции в обработке изображений. Они очень просты и наиболее часто используются в

алгоритмах. Они «полезны» и сами по себе и в сочетании с другими классами алгоритмов обработки изображений. Алгоритм точечных процессов может быть описан как сканирование изображения элемент за элементом и осуществление преобразования элементов изображения.

Применение алгоритмов точечных процессов к изображениям имеет смысл только для полутоновых изображений (grayscale), значения яркости которых представляются значениями элементов изображения. Если же в обработке участвует изображение с форматом пикселя 24 бита на точку (RGB), то обработке подвергается каждый цветовой канал в отдельности.

Гистограмма яркости — диаграмма распределений значений интенсивности элементов в изображении. Гистограмма может свидетельствовать об общей яркости и контрасте изображений. Гистограмма является ценным методом для процесса как количественной, так и качественной обработки изображения. Гистограмма яркости представляет собой массив H_k из 256 элементов. Значение элемента массива H_k есть количество пикселей со значением яркости k .

Пусть обрабатываемое изображение S имеет размеры h строк и w столбцов. Элемент изображения S_{ij} , содержащийся в i -м столбце j -ой строки представляет собой значение интенсивности оттенков серого цвета. Для RGB изображений S_{ij} будет представлять собой тройку байт, содержащих оттенки красного, зеленого и синего цветов. Тогда алгоритм получения гистограммы для изображения в оттенках серого цвета может быть записан следующим образом.

```
for (k = 0; k < 256; k++) {  
    H[k] = 0;  
    for (y = 0; y < h; y++) {  
        for (x = 0; x < w; x++) {  
            if (S[x][y] == k) {  
                H[k]++;  
            }  
        }  
    }  
}
```

}
}

Следует заметить, что $\sum_{k=0}^{255} H_k = w \times h$. Для цветных изображений необходимо задавать уже три массива для хранения значений оттенков красного, зеленого и синего цветов соответственно.

Иногда изображение может быть улучшено с помощью регулировки яркости. Просветление — это точечный процесс, который прибавляет постоянные значения к элементам в изображении. Алгебраически процесс просветления выглядит так: $R_{ij} = S_{ij} + b$, где S_{ij} , R_{ij} — элементы исходного и результирующего изображения соответственно, b — постоянная яркости, которая может быть как положительной, так и отрицательной. Если $b > 0$, то результат обработки — увеличение яркости (просветление), если же $b < 0$, то результат — уменьшение яркости (затемнение). Могут возникнуть случаи, когда результат данного точечного процесса выйдет за границы отрезка $[0, 255]$. В этом случае можно вышедшее за пределы отрезка значение привести к ближайшей границе. В виде формулы это будет выражаться следующим образом.

$$R_{ij} = \begin{cases} S_{ij} + b, & 0 \leq S_{ij} + b \leq 255, \\ 0, & S_{ij} + b < 0, \\ 255, & S_{ij} + b > 255. \end{cases} \quad (2)$$

Графически процесс просветления/затемнения может быть описан с помощью кусочно-линейных функций.

Идея негативного преобразования заключается в том, чтобы сделать часть изображения, которая была светлой — темной, а ту, что была темной — светлой. Негативное изображение создается путем вычитания значения элемента изображения из максимально возможного значения яркости, равного 255. В виде формулы негативное преобразование выражается следующим образом: $R_{ij} = 255 - S_{ij}$. Графически негативный процесс может быть описан с помощью кусочно-линейной функции.

Негативное изображение полезно, если необходимо детально рассмотреть

яркие части изображения. Человеческий глаз гораздо более восприимчив к деталям в темной области изображения, чем в светлой. Иногда вводится некий порог p , чтобы в негатив преобразовывать только самую яркую часть изображения (пиксели, значения, яркости которых превышают заданное значение p).

$$R_{ij} = \begin{cases} 255 - S_{ij}, & S_{ij} \geq p, \\ S_{ij}, & S_{ij} < p. \end{cases} \quad (3)$$

Пороговое преобразование (бинаризация) — метод «превращения» полутонового изображения в изображение, содержащее два цвета: черный — 0 и белый — 255. Математически бинаризация выражается следующим образом:

$$R_{ij} = \begin{cases} 0, & S_{ij} < p, \\ 255, & S_{ij} \geq p. \end{cases} \quad (4)$$

Пороговое преобразование может быть использовано как один из шагов при оцифровке печатных документов. Графически бинаризация выглядит как функция, имеющая форму ступеньки.

Гистограмма интенсивностей яркости является информативным инструментом при обработке изображений. Идеальным методом для проверки контраста изображения являются именно гистограммы. Изображения могут быть сгруппированы в одну из трех категорий контраста. Изображения с *низким* контрастом характеризуются как в основном светлые, либо в основном темные. Изображения с *высоким* контрастом имеют как темные, так и светлые области. Изображения с *нормальным* контрастом используют весь диапазон яркости, соответствующий полутоновым изображениям.

С помощью линейных преобразований над гистограммами интенсивностей выполняется увеличение и уменьшение контраста в изображении. Рассмотрим пример увеличения контраста для гистограммы.

Для гистограмм, соответствующих низкоконтрастным изображениям, выделяют отрезок $[Q_1, Q_2]$, на котором сосредоточена значительная часть интенсивностей. Затем данный отрезок линейно отражают (растягивают) в от-

резок $[0, 255]$ с помощью следующего линейного преобразования:

$$R_{ij} = (S_{ij} - Q_1) \frac{255}{Q_2 - Q_1}. \quad (5)$$

Операцию изменения контраста иногда называют гистограммным растягиванием. Преобразование, при котором контраст уменьшается можно выразить такой формулой:

$$R_{ij} = Q_1 + S_{ij} \frac{Q_2 - Q_1}{255}, \quad (6)$$

где $[Q_1, Q_2]$ — некий отрезок, в который отображается все множество значений интенсивностей. Графически операции увеличения и уменьшения контрастности в изображении можно представить следующим образом.

Цветовая (гамма) коррекция изображения также является примером точечного процесса, при котором значения яркости пикселей изменяются по закону степенных функций. Для осуществления этого преобразования можно воспользоваться следующей формулой

$$R_{ij} = 255 \left(\frac{S_{ij}}{255} \right)^\gamma, \quad (7)$$

где величина γ определяет степень функции преобразования. Как правило, значения γ выбираются из следующих последовательностей:

$$\frac{1}{k} = \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \dots, \quad (8)$$

где $k > 2$ и

$$m = 1, 2, 3, \dots, \quad (9)$$

где $m > 2$.

Описанные методы преобразования обычно используются для изображений в оттенках серого цвета. Полноцветные изображения, содержащие три цветовые канала (красный, зеленый и синий), могут быть подвергнуты преобразованиям путем обработки каждого канала в отдельности.

Задание

Написать программу для обработки растровых изображений. Реализовать в ней все описанные методы. Предусмотреть сохранение результатов.

Контрольные вопросы

1. Дать определение точечных процессов.
2. Перечислить и кратко охарактеризовать методы обработки растровых изображений
3. Дать определение гистограммы яркости. Описать алгоритм ее построения.
4. Записать формулу, описывающую регулировку яркости изображения.
5. Записать формулу, описывающую бинаризацию изображения.
6. Записать формулы, описывающие операцию изменения контраста.

Литература

1. Шикин, Е. В., Боресков, А. В. Компьютерная графика. Полигональные модели [Текст]. — М.: ДИАЛОГ-МИФИ, 2001. — 464 с.
2. Семёнов, А. Б. Обработка и анализ изображений с использованием языка JAVA [Электронный ресурс]. — Sun Microsystems — Россия и СНГ — Материалы курсов, 2007. — Режим доступа: <http://ru.sun.com/research/materials/> Загл. с экрана.
3. Роджерс, Д. Алгоритмические основы машинной графики [Текст]. — М.: Мир, 1989. — 512 с.

7 Растровые преобразования

Цель работы

Целью выполнения данной работы является получение сведений об основных алгоритмах растровых преобразований графических примитивов и навыков по их реализации и применению.

Теоретические сведения

Алгоритм Брезенхема растровой дискретизации отрезка

При построении растрового образа отрезка необходимо, прежде всего, установить критерии «хорошей» аппроксимации. Первое требование состоит в том, что отрезок должен начинаться и кончаться в заданных точках и при этом выглядеть сплошным и прямым (при достаточно высоком разрешении дисплея этого можно добиться). Кроме того, яркость вдоль отрезка должна быть одинаковой и не зависеть от наклона отрезка и его длины. Это требование выполнить сложнее, поскольку горизонтальные и вертикальные отрезки всегда будут ярче наклонных, а постоянная яркость вдоль отрезка опять же достигается на вертикальных, горизонтальных и наклоненных под углом в 45° линиях. И, наконец, алгоритм должен работать быстро. Для этого необходимо по возможности исключить операции с вещественными числами. С целью ускорения работы алгоритма можно также реализовать его на аппаратном уровне.

В большинстве алгоритмов используется пошаговый метод изображения, т. е. для нахождения координат очередной точки растрового образа наращивается значение одной из координат на единицу растра и вычисляется приращение другой координаты.

Задача состоит в построении отрезка, соединяющего на экране точки с координатами (i_1, j_1) , (i_2, j_2) (будем считать, что $i_1 \neq i_2$). Для построения отрезка прямой на плоскости с вещественными координатами можно воспользоваться уравнением прямой, проходящей через две заданные точки, которое

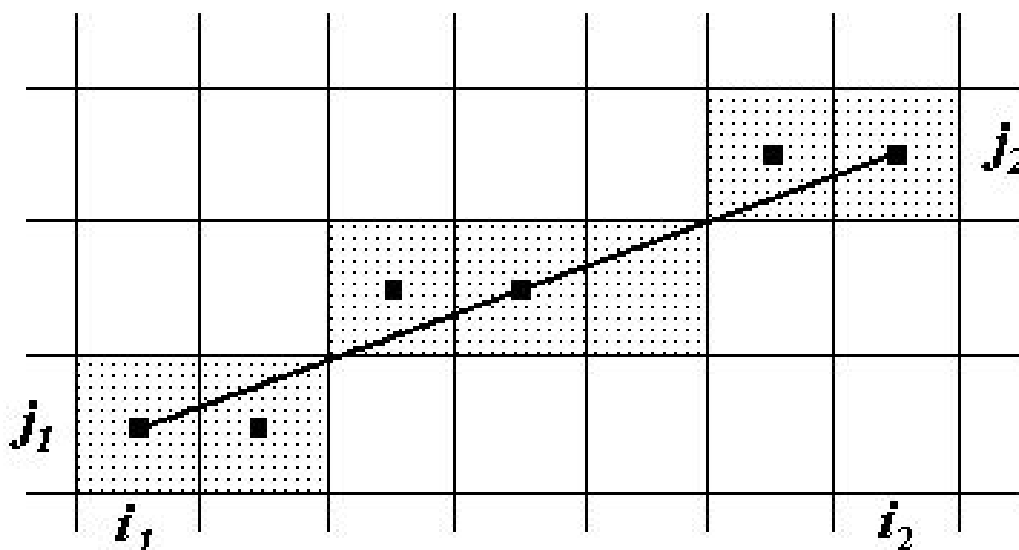


Рис. 6: Растровый образ отрезка.

имеет вид

$$j = j_1 + k(i - i_1), \quad k = (j_2 - j_1)/(i_2 - i_1).$$

Теперь, считая, что $0 \leq k \leq 1$, (i, j) — координаты текущей точки растрового образа, а y — точное значение координаты точки отрезка, можно построить следующую точку:

$$i' = i + 1, \quad j' = y + k.$$

Следует заметить, что целочисленная координата j изменится только в том случае, если y превысит величину $j + 0.5$ (j' есть ближайшее к y целое число, полученное в результате операции округления). Приведенный пример включает операции с вещественными числами, которые выполняются существенно медленнее, чем соответствующие целочисленные операции, а при построении растрового образа отрезка желателен алгоритм, по возможности обращающийся только к целочисленной арифметике. Кроме того, алгоритм должен работать при любом взаимном расположении концов отрезка.

Алгоритм Брезенхема построения растрового образа отрезка был изначально разработан для графопостроителей, но он полностью подходит и для растровых дисплеев. В процессе работы в зависимости от углового коэффициента отрезка наращивается на единицу либо i , либо j , а изменение другой коор-

динаты зависит от расстояния между действительным положением точки и ближайшей точкой раstra (смещения). Алгоритм построен так, что анализируется лишь знак этого смещения.

На рис. 7 это иллюстрируется для отрезка с угловым коэффициентом, лежащим в диапазоне от нуля до единицы. Из рисунка можно заметить, что если угловой коэффициент $k \geq 1/2$, то при выходе из точки $(0, 0)$ пересечение с прямой $x = 1$ будет ближе к прямой $y = 1$, чем к прямой $y = 0$. Следовательно, точка раstra $(1, 1)$ лучше аппроксимирует прохождение отрезка, чем точка $(1, 0)$. При $k < 1/2$ верно обратное.

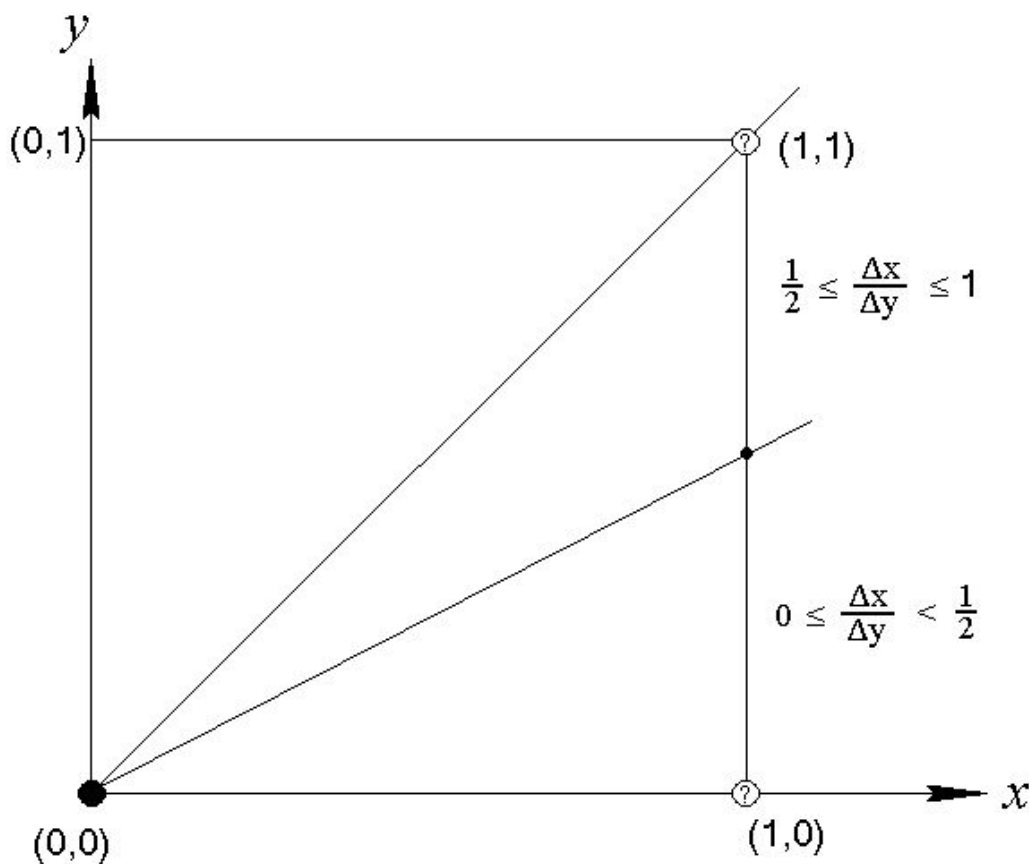


Рис. 7: Связь углового коэффициента с выбором пикселя.

На рис. 8 показано, каким образом строятся точки раstra для отрезка с тангенсом угла наклона $3/8$, а на рис. 9 — график смещения. В начале построения смещение полагается равным $k - 1/2$, а затем на каждом шаге оно наращивается на величину k , и если при этом вертикальная координата точки раstra увеличивается на единицу, то смещение в свою очередь уменьшается

на единицу.

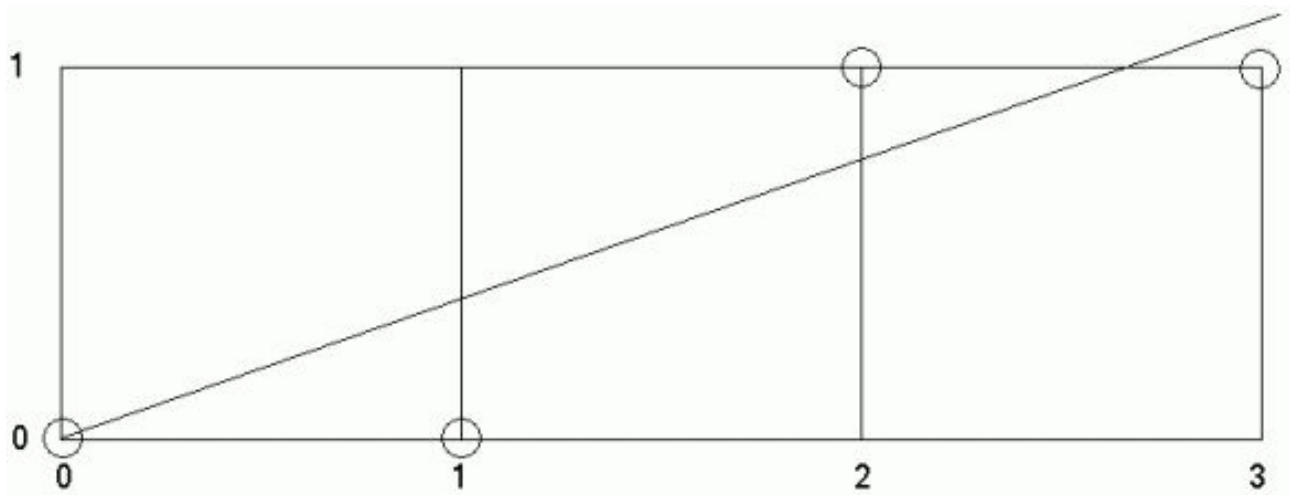


Рис. 8: Пиксели, принадлежащие развертке отрезка.



Рис. 9: График изменения отклонения.

На рис. 10 приведена блок-схема алгоритма для случая $i_2 > i_1$, $j_2 > j_1$, $k > 0$. Нетрудно понять, как от этого алгоритма перейти к целочисленному: достаточно вместо величины смещения e перейти к величине $\tilde{e} = 2\Delta ie$.

Приведем общий алгоритм Брезенхема, который учитывает все возможные случаи направления отрезка, рассматриваемого как вектор на координатной плоскости (на рис. 11 выделены четыре области и указаны особенности алгоритма в каждой из них).

В описании алгоритма используются следующие функции:

swap (a, b): обмен значений переменных a, b; **abs**(a): абсолютное значение a; **sign**(a): 0, если $a = 0$, 1, если $a > 0$, -1, если $a < 0$; **point**(i, j): инициализация точки (i, j).

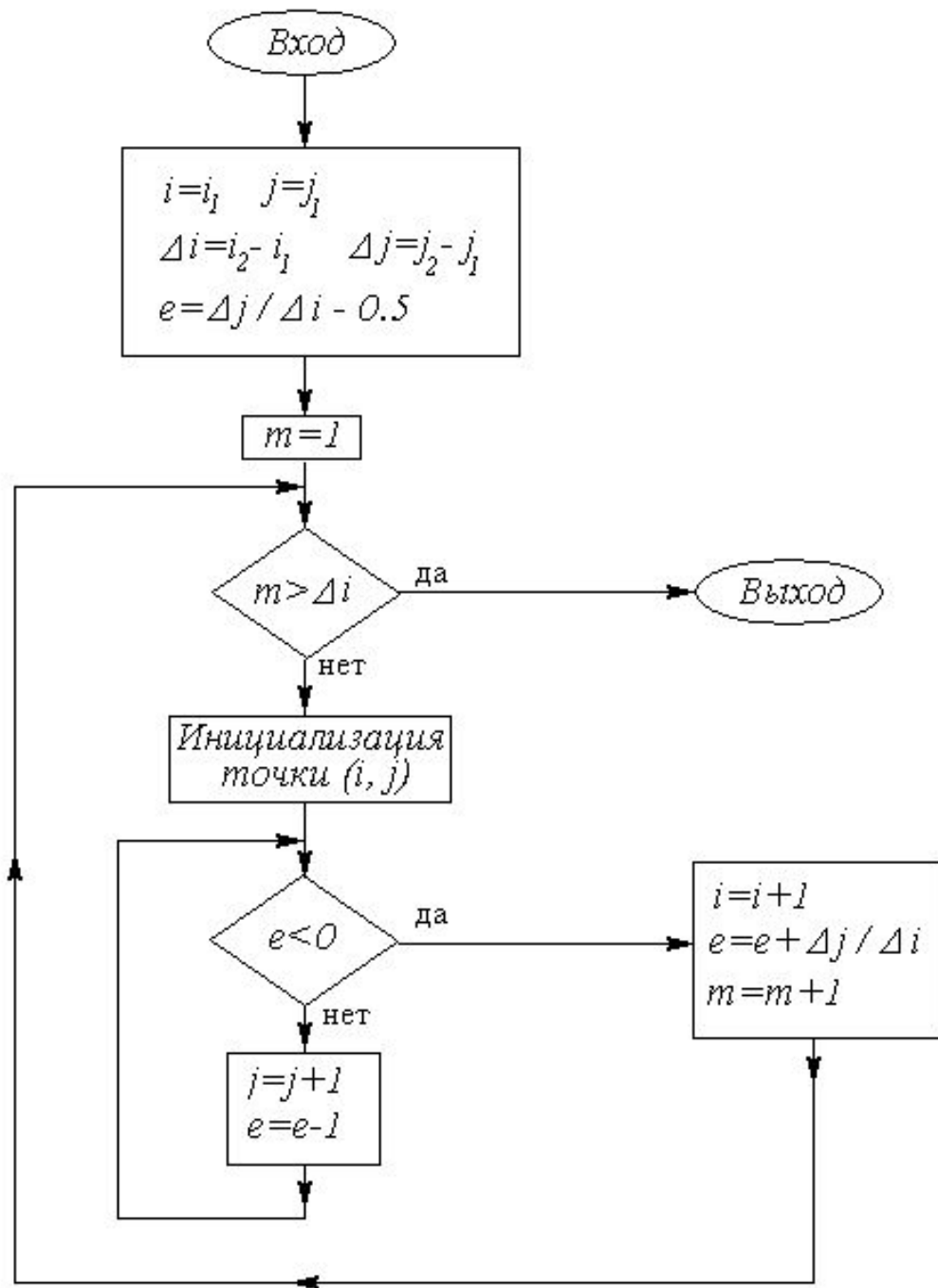


Рис. 10: Блок-схема одной ветви алгоритма Брезенхема.

Предполагается, что концы отрезка (i_1, j_1) , (i_2, j_2) не совпадают и что все используемые переменные являются целыми.

```

i = i1;
j = j1;

```

- I** : *увеличение j на 1*
- II** : *увеличение i на 1*
- III** : *уменьшение j на 1*
- IV** : *уменьшение i на 1*

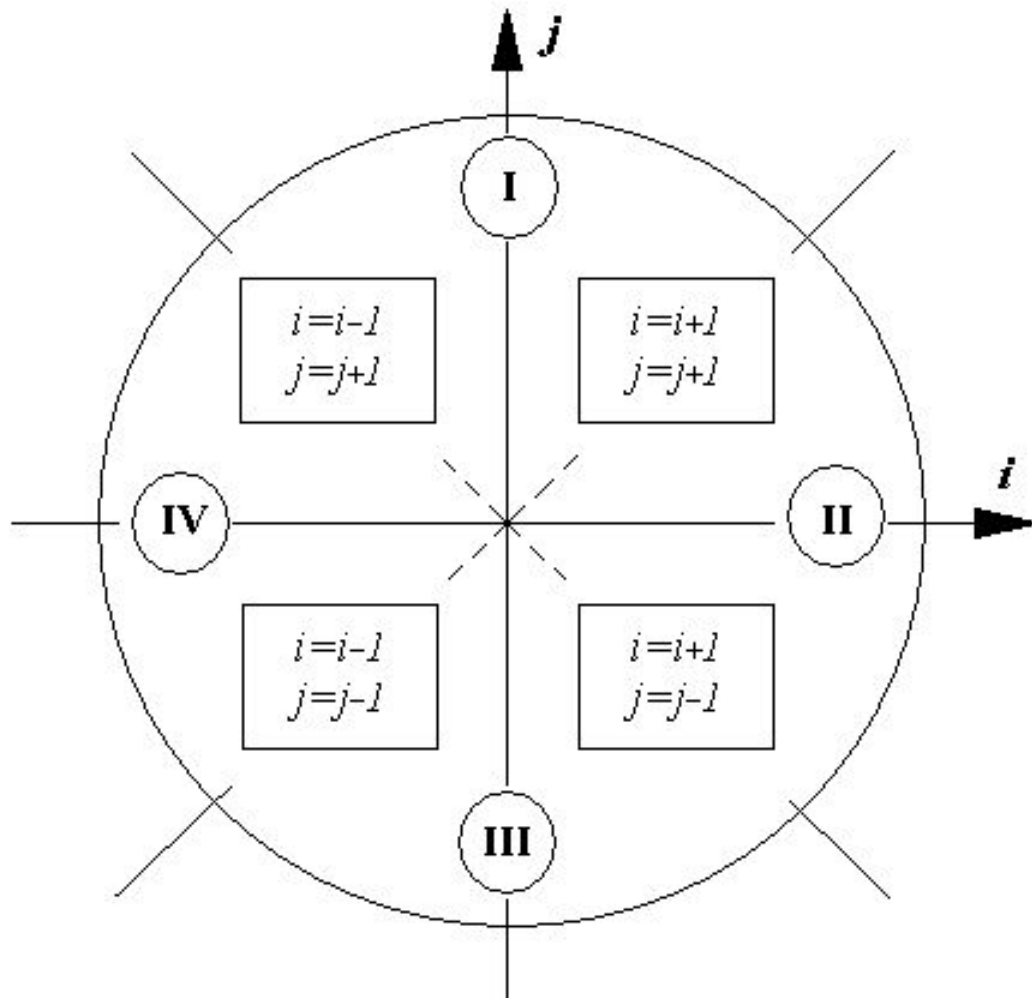


Рис. 11: Четыре возможных направления отрезка.

```

di = i2 - i1;
dj = j2 - j1;
s1 = sign(i2 - i1);
s2 = sign(j2 - j1);
di = abs(di);
dj = abs(dj);
if (dj > di)

```

```

{
    swap(di, dj); c = 1;
}
else c = 0;
e = 2 * dj - di; /* Инициализация смещения */
/* Основной цикл */
for (l = 0; l < di; l++)
{
    point(i, j);
    while (e >= 0) {
        if (c == 1) i = i + s1;
        else j = j + s2;
        e = e - 2*di;
    }
    if (c == 1) j = j + s2;
    else i = i + s1;
    e = e + 2*dj;
}

```

Алгоритмы Брезенхема растровой дискретизации окружности и эллипса

Алгоритм изображения окружности несколько сложнее, чем построение отрезка. Мы рассмотрим его для случая окружности радиуса r с центром в начале координат. Перенесение его на случай произвольного центра не составляет труда. При построении растровой развертки окружности можно воспользоваться ее симметрией относительно координатных осей и прямых $y = \pm x$. Необходимо сгенерировать лишь одну восьмую часть окружности, а остальные ее части можно получить путем отображений симметрии. За основу можно взять часть окружности от 0° до 45° в направлении по часовой стрелке с исходной точкой построения $(r, 0)$. В этом случае координата окружности x является монотонно убывающей функцией координаты y .

При выбранном направлении движения по окружности имеется только три возможности для расположения ближайшего пикселя: на единицу вправо, на единицу вниз и по диагонали вниз (рис. 12). Выбор варианта можно осуществить, вычислив расстояния до этих точек и выбрав минимальное из них:

$$\begin{aligned} d_h &= |s_h|, & d_v &= |s_v|, & d_d &= |s_d|, \\ s_h &= (x+1)^2 + y^2 - r^2, & s_v &= x^2 + (y-1)^2 - r^2, \\ s_d &= (x+1)^2 + (y-1)^2 - r^2. \end{aligned}$$

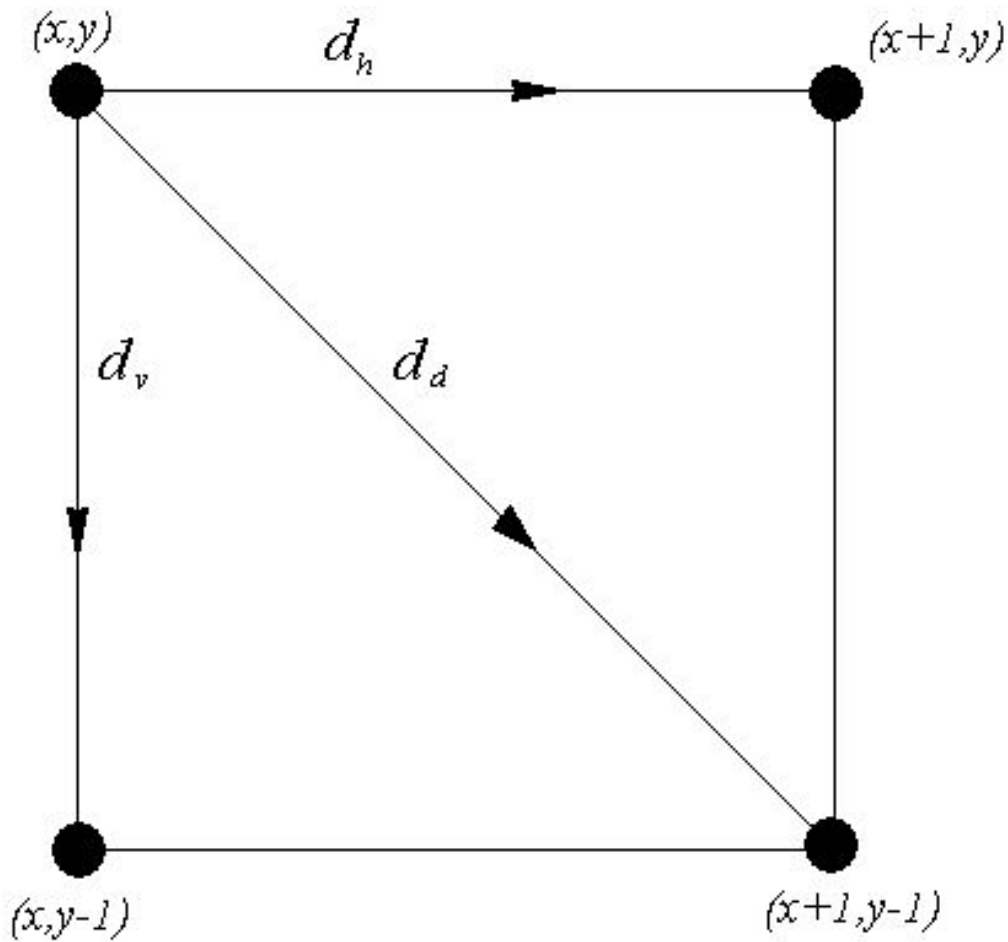


Рис. 12: Ближайший пиксель при движении по окружности.

Алгоритм можно упростить, перейдя к анализу знаков величин s_h , s_v , s_d . При $s_d < 0$ диагональная точка лежит внутри окружности, поэтому ближайшими точками могут быть только диагональная и правая. Теперь достаточно проанализировать знак выражения $\Delta = d_v - d_d$. Если $\Delta \leq 0$, выбираем го-

ризонтальный шаг, в противном случае — диагональный. Если же $s_d > 0$, то определяем знак $\Delta^1 = d_d - d_v$, и если $\Delta^1 \leq 0$, выбираем диагональный шаг, в противном случае — вертикальный. Затем вычисляется новое значение s_d , причем желательно минимизировать вычисления не только этой величины, но и величин Δ , Δ^1 на каждом шаге алгоритма. Путем несложных преобразований можно получить для первого шага алгоритма, что $\Delta = 2(s_d + y) - 1$, $\Delta^1 = 2(s_d + x) - 1$.

После перехода в точку (x', y') , $x' = x + 1$, $y' = y + 1$ по диагонали новое значение s_d вычисляется по формуле $s'_d = s_d + 2x' - 2y' + 2$, при горизонтальном переходе ($x' = x + 1$, $y' = y$) $s'_d = s_d + 2x' + 1$ при вертикальном ($x' = x$, $y' = y + 1$) — $s'_d = s_d + 2y' + 1$.

Таким образом, алгоритм рисования этой части окружности можно считать полностью описанным (блок-схема его приведена на рис. 13). Все оставшиеся ее части строятся параллельно: после получения очередной точки (x', y') можно инициализировать еще семь точек с координатами $(-x', y')$, $(-x', -y')$, $(x', -y')$, (y', x') , $(y', -x')$, $(-y', -x')$, $(-y', x')$.

Для построения растровой развертки эллипса с осями, параллельными осям координат, и радиусами (a, b) воспользуемся каноническим уравнением

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1,$$

которое перепишем в виде

$$f(x, y) = b^2x^2 + a^2y^2 - a^2b^2 = 0.$$

В отличие от окружности, для которой было достаточно построить одну восьмую ее часть, а затем воспользоваться свойствами симметрии, эллипс имеет только две оси симметрии, поэтому придется строить одну четверть всей фигуры. За основу возьмем дугу, лежащую между точками $(0, b)$ и $(a, 0)$ в первом квадранте координатной плоскости.

В каждой точке (x, y) эллипса существует вектор нормали, задаваемый градиентом функции f . Дугу разобьем на две части: первая — с углом между

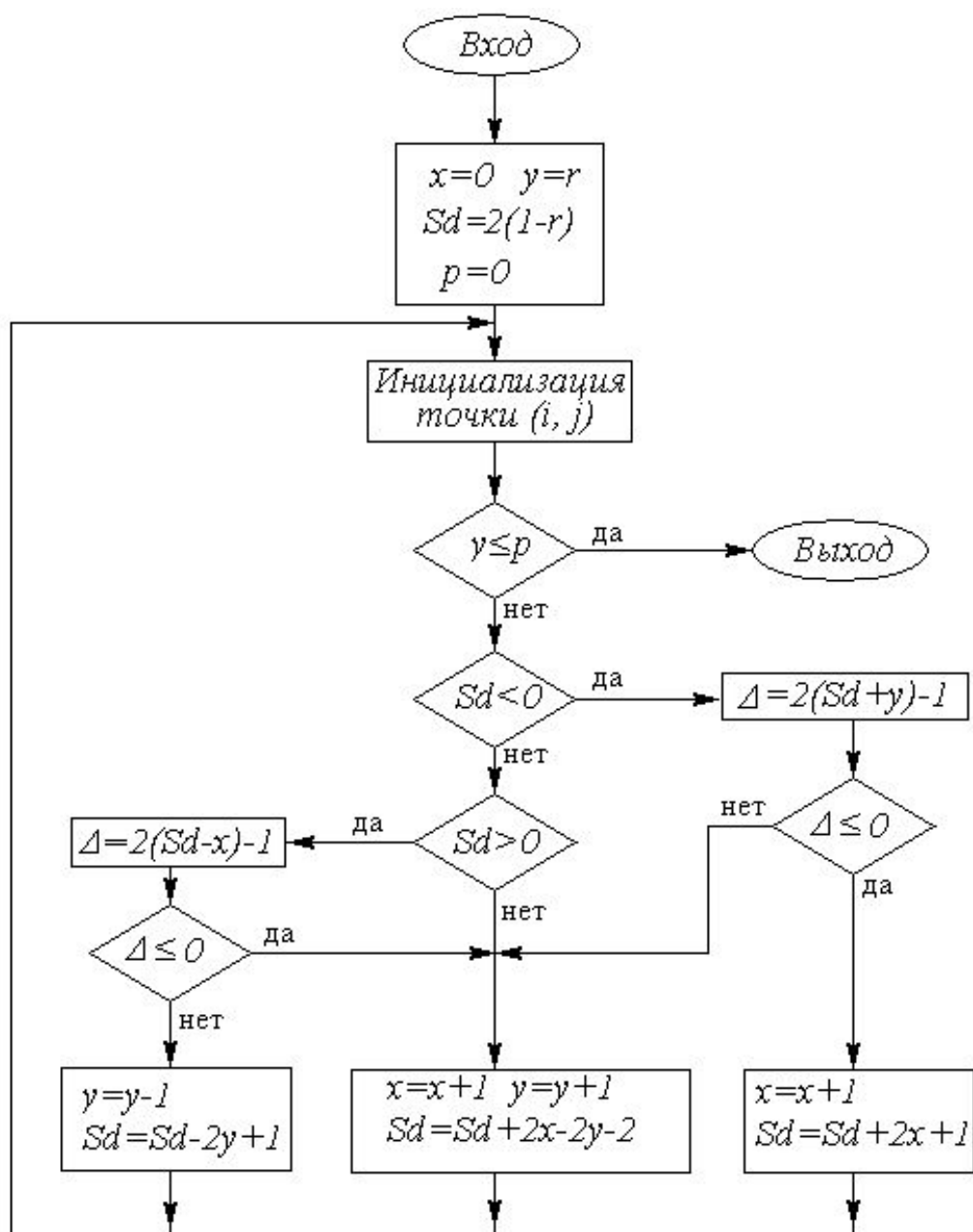


Рис. 13: Блок-схема построения восьмой части окружности.

нормалью и горизонтальной осью больше 45° (тангенс больше 1) и вторая — с углом, меньшим 45° (рис. 14). Движение вдоль дуги будем осуществлять в направлении по часовой стрелке, начиная с точки $(0, b)$. Вдоль всей дуги координата y является монотонно убывающей функцией от x , но в первой части она убывает медленнее, чем растет аргумент, а во второй — быстрее. Поэтому при построении растрового образа в первой части будем увеличивать

x на единицу и искать соответствующее значение y , а во второй — сначала уменьшать значение y на единицу и определять соответствующее значение y .

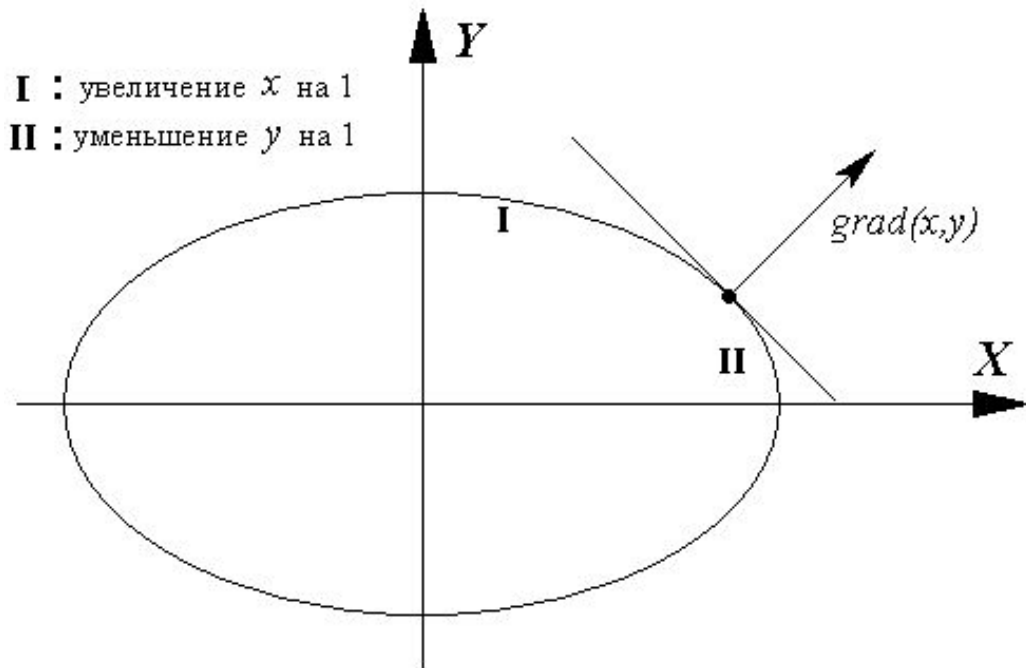


Рис. 14: Две области на участке эллипса.

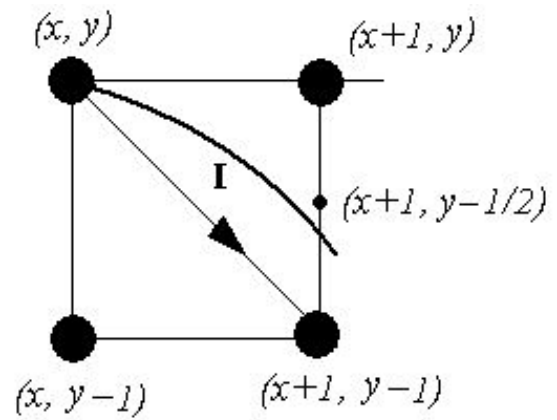
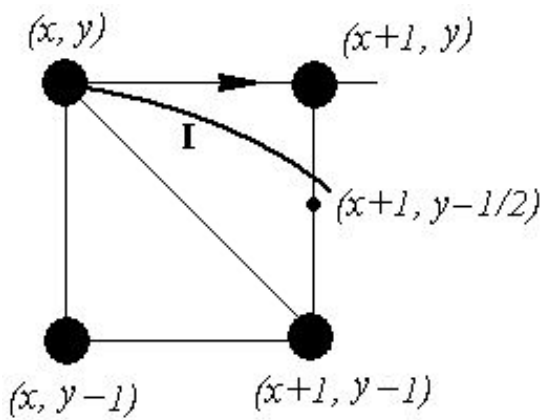
Направление нормали соответствует вектору

$$\text{grad}(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) = (2b^2x, 2a^2y).$$

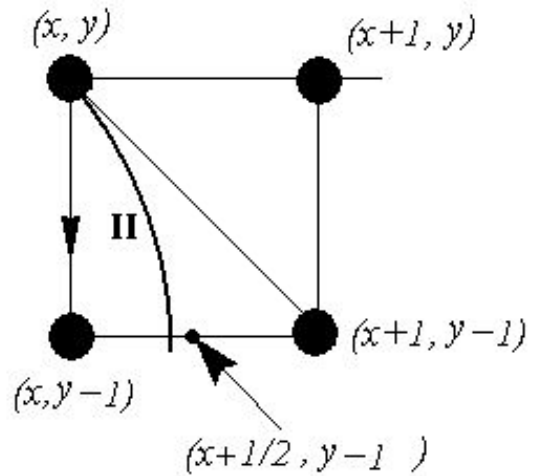
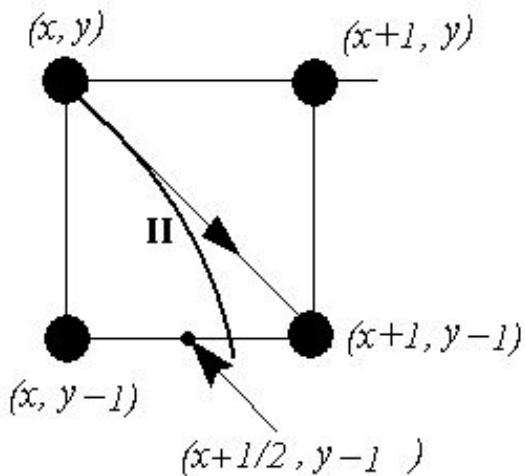
Отсюда находим тангенс угла наклона вектора нормали: $t = a^2y/(b^2x)$. Приравняв его единице, получаем, что координаты точки деления дуги на вышеуказанные части удовлетворяют равенству $b^2x = a^2y$. Поэтому критерием того, что мы переходим ко второй области в целочисленных координатах, будет соотношение $a^2(y - 1/2) \leq b^2(x + 1)$, или, переходя к целочисленным операциям, $a^2(2y - 1) \leq 2b^2(x + 1)$.

При перемещении вдоль первого участка дуги мы из каждой точки переходим либо по горизонтали, либо по диагонали, и критерий такого перехода напоминает тот, который использовался при построении растрового образа окружности. Находясь в точке (x, y) , мы будем вычислять значение $\Delta = f(x + 1, y - 1/2)$. Если это значение меньше нуля, то дополнитель-

ная точка $(x + 1, y - 1/2)$ лежит внутри эллипса, следовательно, ближайшая точка растра есть $(x + 1, y)$, в противном случае это точка $(x + 1, y + 1)$ (рис. 15 а).



а)



б)

Рис. 15: Схема перехода в первой и второй областях дуги эллипса.

На втором участке дуги возможен переход либо по диагонали, либо по вертикали, поэтому здесь сначала значение координаты y уменьшается на единицу, затем вычисляется $\Delta = f(x + 1/2, y - 1)$ и направление перехода выбирается аналогично предыдущему случаю (рис. 15 б).

Остается оптимизировать вычисление параметра Δ , умножив его на 4 и представив в виде функции координат точки. Тогда для первой половины дуги имеем

$$\begin{aligned}\tilde{\Delta}(x, y) &= 4\Delta(x, y) = 4b^2(x + 1)^2 + a^2(2y - 1)^2 - 4a^2b^2, \\ \tilde{\Delta}(x + 1, y) &= 4\tilde{\Delta}(x, y) + 4b^2(2x + 3), \\ \tilde{\Delta}(x + 1, y - 1) &= 4\tilde{\Delta}(x, y) + 4b^2(2x + 3) - 8a^2(y - 1).\end{aligned}$$

Для второй половины дуги получим

$$\begin{aligned}\tilde{\Delta}(x, y) &\equiv 4\Delta(x, y) = b^2(2x + 1)^2 + 4a^2(y - 1)^2 - 4a^2b^2, \\ \tilde{\Delta}(x + 1, y) &= \tilde{\Delta}(x, y) + 8b^2(x + 1), \\ \tilde{\Delta}(x + 1, y - 1) &= \tilde{\Delta}(x, y) + 8b^2(x + 1) - 4a^2(2y + 3).\end{aligned}$$

Все оставшиеся дуги эллипса строятся параллельно: после получения очередной точки (x', y') , можно инициализировать еще три точки с координатами $(-x', y')$, $(-x', -y')$, $(x', -y')$. Блок-схему не приводим ввиду прозрачности алгоритма.

Алгоритмы заполнения областей

Для заполнения областей, ограниченных замкнутой линией, применяются два основных подхода: затравочное заполнение и растровая развертка.

Методы первого типа исходят из того, что задана некоторая точка (затравка) внутри контура и задан критерий принадлежности точки границе области (например, задан цвет границы). В алгоритмах ищут точки, соседние с затравочной и расположенные внутри контура. Если обнаружена соседняя точка, принадлежащая внутренней области контура, то она становится затравочной и поиск продолжается рекурсивно.

Методы растровой развертки основаны на сканировании строк раstra и определении, лежит ли точка внутри заданного контура области. Сканирование осуществляется чаще всего «сверху вниз», а алгоритм определения принадлежности точки заданной области зависит от вида ее границы.

Сначала рассмотрим простой алгоритм заполнения с затравкой с использованием стека. Под стеком в данном случае мы будем понимать массив, в который можно последовательно помещать значения и последовательно извлекать, причем извлекаются элементы не в порядке поступления, а наоборот: по принципу «первым пришел — последним ушел» («first in — last out»). Алгоритм заполнения выглядит следующим образом:

Поместить затравочный пиксель в стек

Пока стек не пуст:

 Извлечь пиксель из стека

 Инициализировать пиксель

 Для каждого из четырех соседних пикселей:

 Проверить, является ли он граничным и был ли он
 инициализирован

 Если нет, то поместить пиксель в стек

Алгоритм можно модифицировать таким образом, что соседними будут считаться восемь пикселей (добавляются элементы, расположенные в диагональном направлении).

Методы растровой развертки рассмотрим сначала в применении к заполнению многоугольников. Простейший метод построения состоит в том, чтобы для каждого пикселя раstra проверить его принадлежность внутренности многоугольника. Но такой перебор слишком неэкономичен, поскольку фигура может занимать лишь незначительную часть экрана, а геометрический поиск — задача трудоемкая, сопряженная с длинными вычислениями. Алгоритм станет более эффективным, если предварительно выявить минимальный прямоугольник, в который погружен контур многоугольника, но и этого может оказаться недостаточно.

В случае, когда многоугольник, ограничивающий область, задан списком вершин и ребер (ребро определяется как пара вершин), можно предложить еще более экономный метод. Для каждой сканирующей строки определяются точки пересечения с ребрами многогранника, которые затем упорядочиваются по координате x . Определение того, какой интервал между парами пересе-

чений есть внутренний для многогранника, а какой нет, является достаточно простой логической задачей. При этом если сканирующая строка проходит через вершину многогранника, то это пересечение должно быть учтено дважды в случае, когда вершина является точкой локального минимума или максимума. Для поиска пересечений сканирующей строки с ребрами можно использовать алгоритм Брезенхема построения растрового образа отрезка.

В заключение в качестве примера приведем алгоритм закрашки внутренней области треугольника, основанный на составлении полного упорядоченного списка всех отрезков, составляющих этот треугольник. Для записи горизонтальных координат концов этих отрезков будем использовать два массива X_{min} и X_{max} размерностью, равной числу пикселей раstra по вертикали (рис. 16).

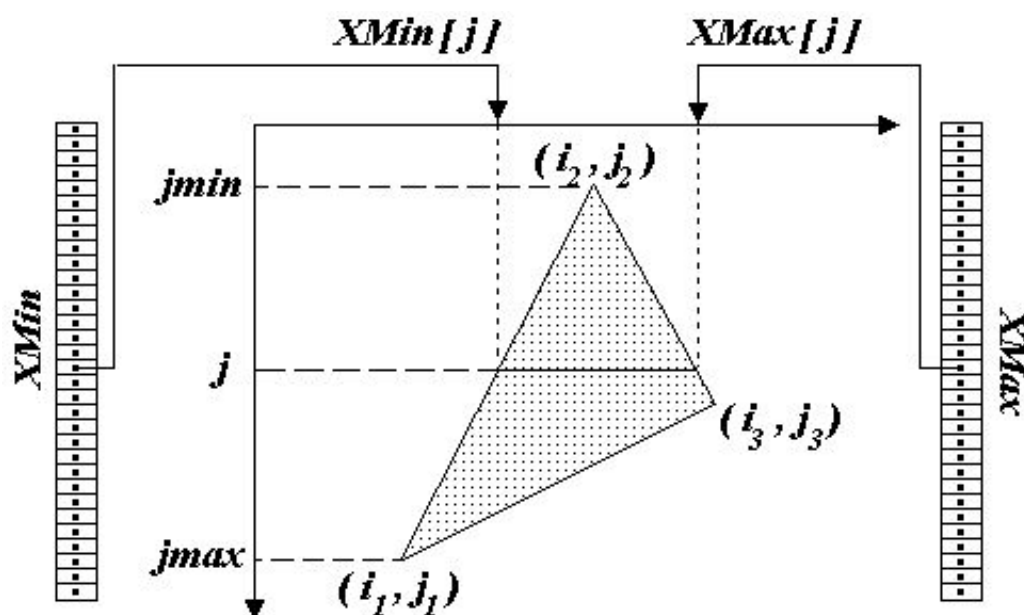


Рис. 16: Схема построения растровой развертки треугольника.

Построение начинается с инициализации массивов X_{min} и X_{max} : массив X_{max} заполняется нулями, а массив X_{min} — числом N , равным числу пикселей раstra по горизонтали. Затем определяем значения j_{min} , j_{max} , ограничивающие треугольник в вертикальном направлении. Теперь, используя модифицированный алгоритм Брезенхема, занесем границы отрезков в массивы X_{min} и X_{max} . Для этого всякий раз при переходе к очередному пикселю

при формировании отрезка вместо его инициализации будем сравнивать его координату i с содержимым j -й ячейки массивов. Если $X_{min}[j] > i$, то записываем координату i в массив X_{min} . Аналогично при условии $X_{max}[j] < i$ координату i записываем в массив X_{max} .

Если теперь последовательно применить алгоритм Брезенхема ко всем трем сторонам треугольника, то мы получим нужным образом заполненные массивы границ. Остается только проинициализировать пиксели внутри отрезков $\{(X_{min}[j], j), (X_{max}[j], j)\}$.

Этот алгоритм можно легко распространить на случай произвольного выпуклого многоугольника.

Задание

Реализовать алгоритмы Брезенхема для построения отрезка и окружности (или эллипса — по указанию преподавателя). Для окружности (эллипса) предварительно доработать алгоритм, учтя возможность любых значений координат центра. Реализовать алгоритм заполнения для равностороннего пятиугольника.

Контрольные вопросы

1. Что такое разложение в растр?
2. Какова математическая основа растрового разложения в алгоритме Брезенхема?
3. По какому критерию инициализируется пиксель в этом алгоритме?
4. Чем отличаются ветви алгоритма при углах наклона $< 45^\circ$ и $> 45^\circ$?
5. Какую часть окружности достаточно построить, чтобы затем путем отражений получить окружность целиком?
6. Какую часть эллипса достаточно построить, чтобы затем путем отражений получить эллипс целиком?

7. Назовите два типа алгоритмов заполнения областей.
8. Какая структура данных используется в алгоритмах с затравкой?
9. Какие данные используются при построении растровой развертки треугольника?

Литература

1. Шикин, Е. В., Боресков, А. В. Компьютерная графика. Полигональные модели [Текст]. — М.: ДИАЛОГ-МИФИ, 2001. — 464 с.
2. Роджерс, Д. Алгоритмические основы машинной графики [Текст]. — М.: Мир, 1989. — 512 с.
3. Куликов, А. И., Овчинникова, Т. Э. Алгоритмические основы современной компьютерной графики [Электронный ресурс]. — Интернет университет информационных технологий intuit.ru, 2007. — Режим доступа: <http://www.intuit.ru/departments/graphics/graphalg/>. — Загл. с экрана.

8 Алгоритмы удаления невидимых линий и поверхностей

Цель работы

Целью выполнения данной работы является изучение основных алгоритмов удаления невидимых линий и поверхностей и получение навыков по их реализации и использованию.

Теоретические сведения

Необходимость удаления невидимых линий, ребер, поверхностей или объемов проиллюстрирована на рис. 17. Рисунок наглядно демонстрирует, что изображение без удаления невидимых линий воспринимается неоднозначно.



Рис. 17: Неоднозначность восприятия изображения куба.

Удаление нелицевых граней многогранника

Алгоритм Робертса. Этот алгоритм, предложенный в 1963 г., является первой разработкой такого рода и предназначен для удаления невидимых линий при штриховом изображении объектов, составленных из выпуклых многогранников. Он относится к алгоритмам, работающим в объектном пространстве, и очень элегантен с математической точки зрения. В нем очень удачно сочетаются геометрические методы и методы линейного программирования.

Выпуклый многогранник однозначно определяется набором плоскостей, образующих его грани, поэтому исходными данными для алгоритма являются многогранники, заданные списком своих граней. Грани задаются в виде

плоскостей, заданных в канонической форме в объектной системе координат:
 $ax + by + cz + d = 0$.

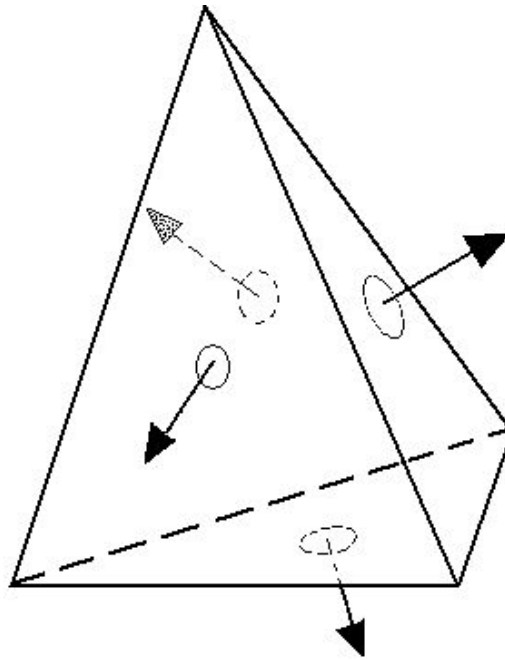


Рис. 18: Внешние нормали тетраэдра.

Таким образом, каждая плоскость определяется четырехмерным вектором \vec{P} , а каждая точка \vec{r} , заданная в однородных координатах, также представляет собой четырехмерный вектор:

$$\vec{P} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}, \vec{r} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

Принадлежность точки плоскости можно установить с помощью скалярного произведения, т.е. если $(\vec{P} \cdot \vec{r}) = 0$, то точка принадлежит плоскости, если же нет, то знак произведения показывает, по какую сторону от плоскости эта точка находится. В алгоритме Робертса плоскости строятся таким образом, что внутренние точки многогранника лежат в положительной полуплоскости. Это означает, что вектор (A, B, C) является внешней нормалью к многограннику (рис. 18). Из векторов плоскостей строится прямоугольная

матрица порядка $4 \times n$, которая называется **обобщенной матрицей описания многогранника**:

$$M = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ b_1 & b_2 & b_3 & \dots & b_n \\ c_1 & c_2 & c_3 & \dots & c_n \\ d_1 & d_2 & d_3 & \dots & d_n \end{pmatrix}.$$

Умножая столбцы матрицы на вектор \vec{r} , получим n -мерный вектор, и если все его компоненты неотрицательны, то точка принадлежит многограннику. Это условие будем записывать в виде $(\vec{r} \cdot M) \geq 0$ (имеется в виду умножение вектор-строки на матрицу).

В своем алгоритме Робертс рассматривает только отрезки, являющиеся пересечением граней многогранника.

Из обобщенной матрицы можно получить информацию о том, какие грани многогранника пересекаются в вершинах. Действительно, если вершина $\vec{v} = (x, y, z, 1)$ принадлежит граням $\vec{P}_1, \vec{P}_2, \vec{P}_3$, то она удовлетворяет уравнениям

$$\left. \begin{aligned} (\vec{v} \cdot \vec{P}_1) &= 0 \\ (\vec{v} \cdot \vec{P}_2) &= 0 \\ (\vec{v} \cdot \vec{P}_3) &= 0 \\ (\vec{v} \cdot \vec{e}_4) &= 1 \end{aligned} \right\}, \quad \text{где} \quad \vec{e}_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Эту систему можно записать в матричном виде:

$$\vec{v} \cdot Q = \vec{e}_4,$$

где Q — матрица, составленная из вектор-столбцов $\vec{P}_1, \vec{P}_2, \vec{P}_3, \vec{e}_4$. Значит, координаты вершины определяются соотношением

$$\vec{v} = \vec{e}_4 \cdot Q^{(-1)},$$

т. е. они составляют последнюю строку обратной матрицы. А это означает,

что если для каких-либо трех плоскостей обратная матрица существует, то плоскости имеют общую вершину.

Алгоритм прежде всего удаляет из каждого многогранника те ребра или грани, которые экранируются самим телом. Робертс использовал для этого простой тест: если одна или обе смежные грани обращены своей внешней поверхностью к наблюдателю, то ребро является видимым. Тест этот выполняется вычислением скалярного произведения координат наблюдателя на вектор внешней нормали грани: если результат отрицательный, то грань видима.

Затем каждое из видимых ребер каждого многогранника сравнивается с каждым из оставшихся многогранников для определения того, какая его часть или части, если таковые есть, экранируются этими телами. Для этого в каждую точку ребра проводится отрезок луча, выходящего из точки расположения наблюдателя. Если отрезок не пересекает ни одного из многогранников, то точка видима. Для решения этой задачи используются параметрические уравнения прямой, содержащей ребро, и луча.

Если заданы концы отрезка \vec{r} и \vec{s} , а наблюдатель расположен в точке \vec{u} , то отрезок задается уравнением

$$\vec{v} = \vec{r} + t \cdot (\vec{s} - \vec{r}) \equiv \vec{r} + t \cdot \vec{d}, \quad 0 \leq t \leq 1,$$

а прямая, идущая в точку, соответствующую параметру t , — уравнением

$$\vec{w} = \vec{v} + \tau \cdot \vec{g} = \vec{r} + t \cdot \vec{d} + \tau \cdot \vec{g}.$$

Для определения той части отрезка, которая закрывается каким-либо телом, достаточно найти значения t и τ , при которых произведение вектора \vec{w} на обобщенную матрицу положительно. Для каждой плоскости \vec{P}_i записывается неравенство

$$q_i = (\vec{v} \cdot \vec{P}_i) + t(\vec{d} \cdot \vec{P}_i) + \tau(\vec{g} \cdot \vec{P}_i) > 0.$$

Эти условия должны выполняться для всех плоскостей.

Полагая $q_i = 0$, получаем систему уравнений, решения которой дают нам точки «смены видимости» отрезка. Результат можно получить путем совместного решения всевозможных пар уравнений из этой системы. Число всевозможных решений при N плоскостях равно $N \cdot (N - 1)/2$.

Так как объем вычислений растет с увеличением числа многоугольников, то желательно по мере возможности сокращать их число, т.е. если мы аппроксимируем некоторую поверхность многогранником, то в качестве граней можно использовать не треугольники, а более сложные многоугольники. При этом, разумеется, встает проблема, как построить такой многоугольник, чтобы он мало отклонялся от плоской фигуры.

Алгоритм Варнока. В отличие от алгоритма Робертса, Варнок в 1968 г. предложил алгоритм, работающий не в объектном пространстве, а в пространстве образа. Он также нацелен на изображение многогранников, а главная идея его основана на гипотезе о способе обработки информации, содержащейся в сцене, глазом и мозгом человека. Эта гипотеза заключается в том, что тратится очень мало времени и усилий на обработку тех областей, которые содержат мало информации. Большая часть времени и труда затрачивается на области с высоким информационным содержанием. Так, например, рассматривая помещение, в котором имеется только картина на стене, мы быстро осматриваем стены, пол и потолок, а затем все внимание сосредотачиваем на картине. В свою очередь, на этой картине, если это портрет, мы бегло отмечаем фон, а затем более внимательно рассматриваем лицо изображенного персонажа, в особенности глаза, губы. Как правило, достаточно детально рассматриваются еще и руки и с чуть меньшим вниманием — одежда.

В алгоритме Варнока и его вариантах делается попытка воспользоваться тем, что большие области изображения однородны. Такое свойство называют **когерентностью**, имея в виду, что смежные области (пиксели) вдоль обеих осей и имеют тенденцию к однородности.

В пространстве изображения рассматривается окно и решается вопрос о том, пусто ли оно, или его содержимое достаточно просто для визуализации.

Если это не так, то окно разбивается на фрагменты до тех пор, пока содержимое фрагмента не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения. В последнем случае информация, содержащаяся в окне, усредняется, и результат изображается с одинаковой интенсивностью или цветом.

Конкретная реализация алгоритма Варнока зависит от метода разбиения окна и от деталей критерия, используемого для того, чтобы решить, является ли одержимое окна достаточно простым. В оригинальной версии алгоритма каждое окно разбивалось на четыре одинаковых подокна. Многоугольник, входящий в изображаемую сцену, по отношению к окну будем называть (рис. 19)

- **внешним**, если он целиком находится вне окна;
- **внутренним**, если он целиком расположен внутри окна;
- **пересекающим**, если он пересекает границу окна;
- **охватывающим**, если окно целиком расположено внутри него.

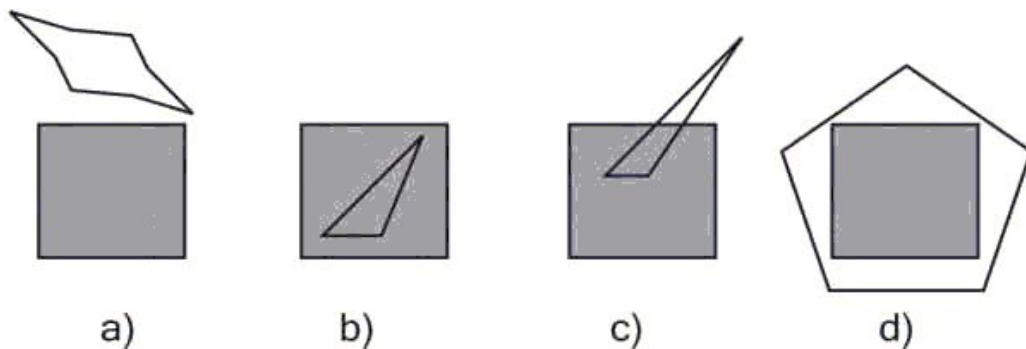


Рис. 19: Варианты расположения многоугольника по отношению к окну.

Теперь можно в самом общем виде описать алгоритм.

Для каждого окна:

1. Если все многоугольники сцены являются внешними по отношению к окну, то оно пусто; изображается фоновым цветом и дальнейшему разбиению не подлежит.

2. Если только один многоугольник сцены имеет общие точки с окном и является по отношению к нему внутренним, то окно заполняется фоновым цветом, а сам многоугольник заполняется своим цветом.
3. Если только один многоугольник сцены имеет общие точки с окном и является по отношению к нему пересекающим, то окно заполняется фоновым цветом, а часть многоугольника, принадлежащая окну, заполняется цветом многоугольника.
4. Если только один многоугольник охватывает окно и нет других многоугольников, имеющих общие точки с окном, то окно заполняется цветом этого многоугольника.
5. Если существует хотя бы один многоугольник, охватывающий окно, то среди всех таких многоугольников выбирается тот, который расположен ближе всех многоугольников к точке наблюдения, и окно заполняется цветом этого многоугольника.
6. В противном случае производится новое разбиение окна.

Шаги 1—4 рассматривают ситуацию пересечения окна только с одним многоугольником. Они используются для сокращения числа подразбиений. Шаг 5 решает задачу удаления невидимых поверхностей. Многоугольник, находящийся ближе всех к точке наблюдения, экранирует все остальные.

Для реализации алгоритма необходимы функции, определяющие взаимное расположение окна и многоугольника, которые достаточно легко реализуются в случае прямоугольных окон и выпуклых многоугольников. Для определения, является ли многоугольник охватывающим, внешним или внутренним, можно воспользоваться, например, погружением многоугольника в прямоугольную оболочку. Для определения наличия пересечений можно использовать опорные прямые (так же, как использовались плоскости в алгоритме Робертса). Если же многоугольник невыпуклый, то задача усложняется. Методы решения такого рода задач здесь мы рассматривать не будем.

Следует заметить, что существуют различные реализации алгоритма Варнока. Были предложены варианты оптимизации, использующие предварительную сортировку многоугольников по глубине, т. е. по расстоянию от точки наблюдения, и другие.

Алгоритм Вейлера—Азертона. Вейлер и Азертон попытались оптимизировать алгоритм Варнока в отношении числа выполняемых разбиений, перейдя от прямоугольных разбиений к разбиениям вдоль границ многоугольников (1977). Для этого они использовали ими же разработанный алгоритм отсечения многоугольников. Алгоритм работает в объектном пространстве, и результатом его работы являются многоугольники. В самом общем виде он состоит из четырех шагов.

1. Предварительная сортировка по глубине.
2. Отсечение по границе ближайшего к точке наблюдения многоугольника, называемое сортировкой многоугольников на плоскости.
3. Удаление многоугольников, экранируемых более близкими к точке наблюдения многоугольниками.
4. Если требуется, то рекурсивное разбиение и новая сортировка.

В процессе предварительной сортировки создается список приблизительных приоритетов, причем близость многоугольника к точке наблюдения определяется расстоянием до ближайшей к ней вершины. Затем выполняется отсечение по самому первому из многоугольников. Отсечению подвергаются все многоугольники из списка, причем эта операция выполняется над проекциями многоугольников на картинную плоскость. При этом создаются списки внешних и внутренних фигур. Все попавшие в список внешних не экранируются отсекающим многоугольником. Затем рассматривается список внутренних многоугольников и выполняется сортировка по расстоянию до отсекающего многоугольника. Если все вершины некоторого многоугольника

оказываются дальше от наблюдателя, чем самая удаленная из вершин экранирующего, то они невидимы, и тогда они удаляются. После этого работа алгоритма продолжается с внешним списком.

Если какая-то из вершин внутреннего многоугольника оказывается ближе к наблюдателю, чем ближайшая из вершин экранирующего многоугольника, то такой многоугольник является частично видимым. В этом случае предварительный список приоритетов некорректен, и тогда в качестве нового отсекающего многоугольника выбирается именно этот «нарушитель порядка». При этом используется именно исходный многоугольник, а не тот, что получился в результате первого отсечения. Такой подход позволяет минимизировать число разбиений.

Этот алгоритм в дальнейшем был обобщен Кэтмулом (1974) для изображения гладких бикубических поверхностей. Его подход заключался в том, что разбиению подвергалась поверхность. Коротко этот алгоритм можно описать так:

1. Рекурсивно разбивается поверхность до тех пор, пока проекция элемента на плоскость изображения не будет покрывать не больше одного пикселя.
2. Определить атрибуты поверхности в этом пикселе и изобразить его.

Эффективность такого метода, как и алгоритм Варнока, зависит от эффективности разбиений. В дальнейшем этот алгоритм был распространен на сплайновые поверхности.

Метод Z-буфера

Это один из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Кэтмулом в 1975 г. Работает этот алгоритм в пространстве изображения. Идея Z-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов каждого пикселя в пространстве изображения, а Z-буфер предназначен для запоминания глубины (расстояния от картинной плоскости) каждого видимого

пикселя в пространстве изображения. Поскольку достаточно распространенным является использование координатной плоскости в качестве картинной плоскости, то глубина равна z -координате точки, отсюда и название буфера. В процессе работы значение глубины каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в Z -буфер. Если это сравнение показывает, что новый пиксель расположен впереди пикселя, находящегося в буфере кадра, то новый пиксель заносится в этот буфер и, кроме того, производится корректировка Z -буфера новым значением глубины. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по x и y наибольшего значения функции $z(x, y)$.

Главное преимущество алгоритма — его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей. Сцены могут быть любой сложности. Поскольку габариты пространства изображения фиксированы, оценка вычислительной трудоемкости алгоритма не более чем линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в Z -буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины. Поэтому экономится вычислительное время, затрачиваемое на сортировку по глубине.

Основной недостаток алгоритма — большой объем требуемой памяти. В последнее время в связи с быстрым ростом возможностей вычислительной техники этот недостаток становится менее лимитирующим. Но в то время, когда алгоритм еще только появился, приходилось изобретать способы создания буфера как можно большего объема при имеющемся ресурсе памяти.

Например, можно разбивать пространство изображения на 4, 16 или больше прямоугольников или полос. В предельном варианте можно использовать буфер размером в одну строку развертки. Для последнего случая был разработан алгоритм построчного сканирования. Поскольку каждый элемент сцены обрабатывается много раз, то сегментирование Z -буфера, вообще говоря, приводит к увеличению времени, необходимого для обработки сцены.

Другой недостаток алгоритма состоит в трудоемкости реализации эффектов, связанных с полупрозрачностью, и ряда других специальных задач, повышающих реалистичность изображения. Поскольку алгоритм заносит пиксели в буфер кадра в произвольном порядке, то довольно сложно получить информацию, которая необходима для методов, основывающихся на предварительном анализе сцены.

В целом алгоритм выглядит так:

1. Заполнить буфер кадра фоновым значением цвета.
2. Заполнить Z -буфер минимальным значением z (глубины).
3. Преобразовать изображаемые объекты в растровую форму в произвольном порядке.
4. Для каждого объекта:
 - Для каждого пикселя (x, y) образа вычислить его глубину $z(x, y)$.
 - Сравнить глубину $z(x, y)$ со значением глубины, хранящимся в Z -буфере в этой же позиции.
 - Если $z(x, y) > Z - \text{буфер}(x, y)$, то занести атрибуты пикселя в буфер кадра и заменить $Z - \text{буфер}(x, y)$ на $z(x, y)$. В противном случае никаких действий не производить.

Алгоритм, использующий Z -буфер, можно также применять для построения сечений поверхностей. Изменится только оператор сравнения:

$$z(x, y) > Z - \text{буфер}(x, y) \text{ и } z(x, y) = z \text{ сечения,}$$

где z сечения — глубина искомого сечения.

Методы приоритетов

Здесь мы рассмотрим группу методов, учитывающих специфику изображаемой сцены для удаления невидимых линий и поверхностей.

При изображении сцен со сплошным закрашиванием поверхностей можно воспользоваться методом художника: элементы сцены изображаются в последовательности от наиболее удаленных от наблюдателя к более близким. При экранировании одних участков сцены другими невидимые участки просто закрашиваются. Если вычислительная трудоемкость получения изображения для отдельных элементов достаточно высока, то такой алгоритм будет не самым лучшим по эффективности, но зато мы избежим анализа (и вполне возможно, тоже дорогостоящего), позволяющего установить, какие же из элементов изображать не надо в силу их невидимости. Например, при изображении правильного многогранника мы довольно легко можем упорядочить его грани по глубине, но такая сортировка для произвольного многогранника возможна далеко не всегда. Мы рассмотрим применение этого метода на примере изображения поверхности, заданной в виде однозначной функции двух переменных.

Пусть поверхность задана уравнением

$$z = f(x, y), \quad a \leq x \leq b, \quad c \leq y \leq d.$$

В качестве картинной плоскости выберем плоскость XOY . В области задания функции на осях координат построим сетку узлов:

$$a = x_0 < x_1 < \dots < x_{n-1} = b, \quad c = y_0 < y_1 < \dots < y_{m-1} < y_m = d.$$

Тогда $z_{ij} = f(x_i, y_j)$ представляют собой набор «высот» для данной поверхности по отношению к плоскости XOY . Поверхность будем аппроксимировать треугольниками с вершинами в точках $\vec{r}_{ij} = (x_i, y_j, z_{ij})$ так, что каждому прямоугольнику сетки узлов будут соответствовать два треугольника: $tr_{ij}^1 = \{\vec{r}_{ij}, \vec{r}_{i+1j}, \vec{r}_{i+1j+1}\}$ и $tr_{ij}^2 = \{\vec{r}_{ij}, \vec{r}_{ij+1}, \vec{r}_{i+1j+1}\}$. Для построения наглядного изображения поверхности повернем ее на некоторый угол сначала относительно оси OX , а затем относительно оси OY , причем направление вращения выберем таким образом, что точки, соответствующие углам координатной сетки, расположатся в следующем порядке по удаленности от

картинной плоскости: \vec{r}_{nm} , \vec{r}_{n0} , \vec{r}_{0m} , \vec{r}_{00} , т. е. точка \vec{r}_{nm} окажется наиболее близкой к картинной плоскости (и наиболее удаленной от наблюдателя). Предполагается, что способ закрашивания треугольников уже определен. Тогда процесс изображения поверхности можно коротко записать так:

Для $i = n, \dots, 1$

Для $j = m, \dots, 1$

Нарисовать tr_{ij}^1 ; нарисовать tr_{ij}^2 .

При такой последовательности вывода изображения мы продвигаемся от самого удаленного треугольника к все более близким, частично закрашивая уже изображенные участки поверхности.

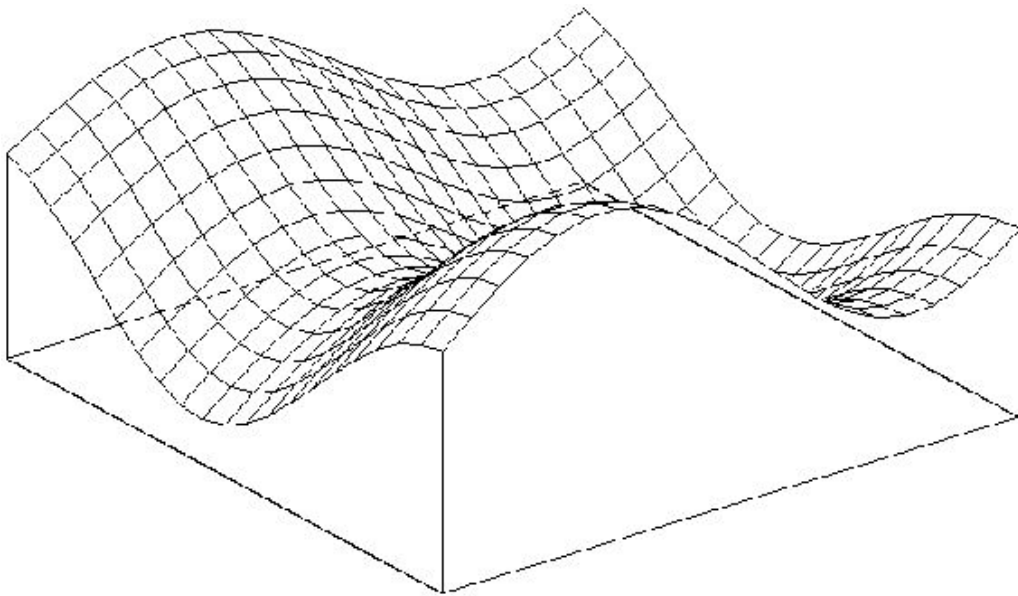


Рис. 20: Простое каркасное изображение с поверхности.

Алгоритм художника можно применять для полностью закрашенной сцены, а для каркасного изображения, когда объект представляется в виде набора кривых или ломаных линий, он непригоден. Для этого случая предложен еще один метод, весьма эффективный — метод плавающего горизонта. Вернемся к предыдущему примеру изображения поверхности. Каркасное изображение получается путем изображения кривых, получаемых при пересечении этой поверхности плоскостями $x = x_i$ и $y = y_i$ (рис. 20).

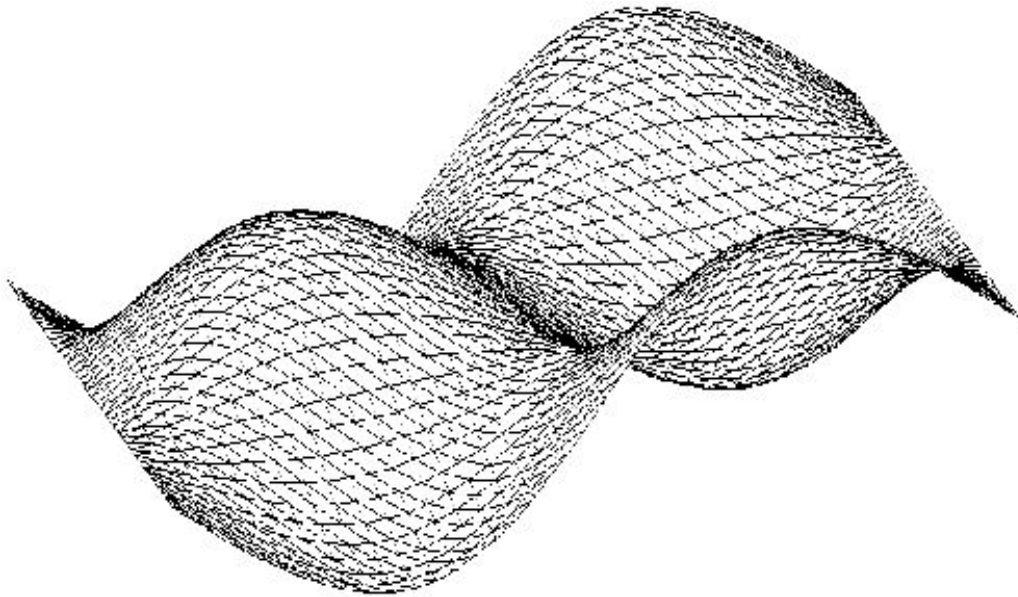


Рис. 21: Каркасное изображение диагональными ребрами.

На самом деле мы будем рисовать четырехугольник и одну диагональ. В процессе рисования нам понадобятся два целочисленных массива: *LHor* (нижний горизонт) и *HHor* (верхний горизонт) размерностью, соответствующей горизонтальному размеру экрана в пикселях. Они нужны для анализа видимости участков изображаемых отрезков. Сначала мы инициализируем верхний горизонт нулем, а нижний — максимальным значением вертикальной координаты на экране. Каждая выводимая на экран точка может закрывать другие точки, которые «скрываются за горизонтом». По мере рисования нижний горизонт «опускается», а верхний «поднимается», постепенно оставляя все меньше незакрытого пространства. В отличие от метода художника, здесь мы продвигаемся от ближнего угла к дальнему. Теперь опишем алгоритм подробнее.

Функция *segment* в этом фрагменте предназначена для вывода на экран отрезка прямой, причем в момент инициализации очередного пикселя она выполняет следующие действия:

Если $(HHor[i] < j)$, то $HHor[i] = j$; вывести пиксель;

Иначе если $(LHor[i] > j)$, то $LHor[i] = j$; вывести пиксель.

Таким образом, пиксель выводится только в том случае, если он выше верхнего или ниже нижнего горизонта, после чего его координаты уже сами становятся одним из горизонтов. А в целом алгоритм будет выглядеть так:

Для $i = 0, \dots, n - 1$

Для $j = 0, \dots, m - 1$

$segment(x_i, y_i, x_{i+1}, y_j); segment(x_i, y_i, x_{i+1}, y_{j+1});$

$segment(x_i, y_i, x_i, y_{j+1}).$

На рис. 21 приведен пример изображения поверхности с использованием этого алгоритма.

Алгоритмы построчного сканирования для криволинейных поверхностей

Идея построчного сканирования, предложенная в 1967 г. Уайли, Ромни, Эвансом и Эрдалом, заключалась в том, что сцена обрабатывается в порядке прохождения сканирующей прямой. В объектном пространстве это соответствует проведению секущей плоскости, перпендикулярной пространству изображения. Строго говоря, алгоритм работает именно в пространстве изображения, отыскивая точки пересечения сканирующей прямой с ребрами многоугольников, составляющих картину (для случая изображения многогранников). Но при пересечении очередного элемента рисунка выполняется анализ глубины полученной точки и сравнение ее с глубиной других точек на сканирующей плоскости. В некоторых случаях можно построить список ребер, упорядоченный по глубине, но зачастую это невозможно выполнить корректно. Поэтому сканирование сочетают с другими методами.

Один из таких методов мы уже рассматривали — метод Z-буфера, в котором буфер инициализируется заново для каждой сканирующей строки. Другой метод называют интервальным алгоритмом построчного сканирования. В нем сканирующая строка разбивается проекциями точек пересечения ребер многоугольников на интервалы, затем в каждом из интервалов выбираются

видимые отрезки. В этой ситуации их уже можно отсортировать по глубине. Мы остановимся чуть подробнее на методе построения сканирования для криволинейных поверхностей.

В описании метода приоритетов поверхность задавалась в виде функции двух переменных, здесь мы будем задавать поверхность параметрическими уравнениями:

$$x = x(u, v), \quad y = y(u, v), \quad z = z(u, v).$$

Пересечение сканирующей плоскости $y = y_1$ с поверхностью дает нам так называемую **линию уровня**, или **изолинию**. Эта кривая может быть неодносвязной, т. е. состоять из нескольких отдельных кривых. Чтобы получить эту кривую, мы должны решить уравнение

$$y(u, v) = y_i$$

и, определив значения параметров u, v , найти точки кривой. Для получения решения можно воспользоваться численными итерационными методами, но это вносит дополнительные проблемы (например, при плохом выборе начального приближения итерационный процесс может не сойтись). Выбор подходящего метода решения лежит вне задач нашего курса, поэтому перейдем к описанию алгоритма, считая, что он уже выбран и надежно работает.

Для каждой сканирующей строки со значением ординаты y_i :

Для каждого значения абсциссы x_j :

Для всех решений уравнений $u = u(x_j, y_j)$, $v = v(x_j, y_j)$ вычислить глубину $z = z(u, v)$.

Определить точку с наименьшим значением глубины и изобразить.

Второй шаг этого алгоритма предполагает, что решение отыскивается только для тех элементов поверхности (или группы поверхностей, если речь идет о более сложной сцене, содержащей составные объекты), которые при данном

значении ординаты пересекают сканирующую плоскость. Предварительный анализ в некоторых случаях позволяет оптимизировать алгоритм.

Метод двоичного разбиения пространства

Теперь разберем один способ использования метода художника при изображении пространственных сцен, содержащих несколько объектов или составные объекты. Это так называемый метод двоичного разбиения пространства плоскостями. Плоскости, как обычно, будут задаваться с помощью вектора нормали \vec{n} и расстояния до начала координат d (с точностью до знака).

Пусть изображаемая сцена состоит из набора непересекающихся граней F_1, F_2, \dots, F_n (они могут иметь общие прямолинейные участки границы). Проведем плоскость P_1 , разбивающую все пространство на два полупространства, в одном из которых находится наблюдатель. Предположим, что плоскость при этом не пересекает ни одну из граней (но может содержать участок ее границы). Тогда грани, находящиеся в одном полупространстве с наблюдателем, могут заслонять от него часть граней из второго полупространства, но не наоборот. Это означает, что они должны изображаться позже.

Разобьем плоскостью P_2 второе полупространство и снова определим, какая группа граней из него должна изображаться раньше. Продолжая этот процесс до того уровня, когда все пространство будет разбито плоскостями на секции, в каждой из которых будет находиться только одна грань, мы получим упорядоченный набор граней.

Этот порядок можно изобразить в виде двоичного дерева. В контексте рассматриваемого алгоритма это дерево представляет собой структуру данных T , элементами которой являются указатель на грань изображаемой сцены, плоскость, отделяющая эту грань, указатели на левое и правое поддеревья TL и TR . Такой элемент называется узлом дерева.

В каждом узле дерева левое поддерево будет содержать грани, отделенные плоскостью, а правое — не отделенные. Рисование сцены осуществляется с

помощью рекурсивного алгоритма следующего вида:

Рисуем дерево (T) :

Если наблюдатель находится в положительной полуплоскости, то:

Если правое поддерево TR не пусто, рисуем дерево (TR).

Рисуем корневую грань.

Если левое поддерево TL не пусто, рисуем дерево (TL).

Иначе

Если левое поддерево TL не пусто, рисуем дерево (TL).

Рисуем корневую грань.

Если правое поддерево TR не пусто, рисуем дерево (TR).

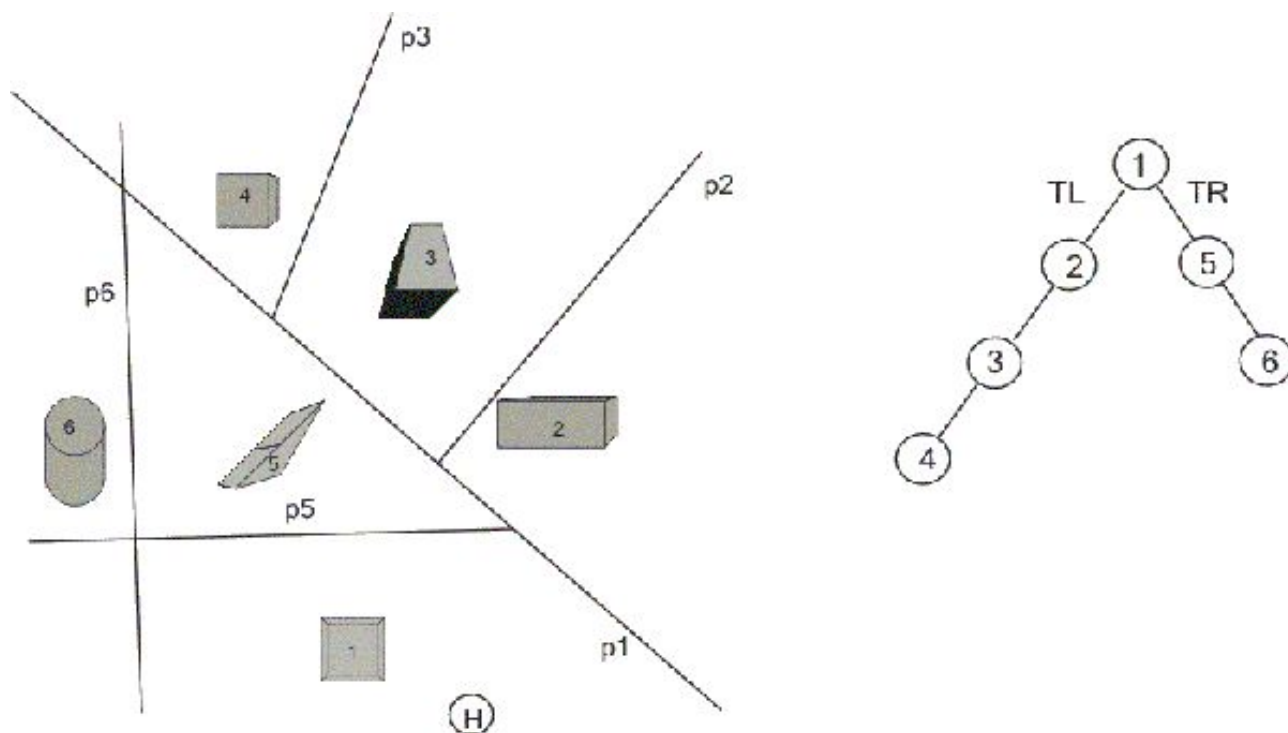


Рис. 22: Разбиение пространства и соответствующее ему дерево.

Построение плоскостей и дерева в данном случае осуществляется «вручную». Для эффективности работы алгоритма надо стремиться к тому, чтобы дерево было сбалансированным. Если какие-то грани не удастся отделить, то их пересекают плоскостями и рисуют как два объекта. Способ опреде-

ления, по какую сторону плоскости находится наблюдатель, а по какую — грань, очень прост. Параметр плоскости d для каждой грани будем задавать так, чтобы грань находилась в положительной полуплоскости. Тогда если при подстановке координат наблюдателя в это уравнение получаем положительное значение, то он находится в одной полуплоскости с гранью, если нет, то в разных.

Алгоритм может применяться не только к многогранникам, но и вообще к любой сцене при условии, что имеется алгоритм изображения составляющих ее объектов. На рис. 22 изображена проекция сцены, разбитой вертикальными плоскостями, и соответствующее ей дерево. Положение наблюдателя отмечено кружком с буквой Н. При этой точке зрения объекты будут изображаться в последовательности 5, 6, 1, 2, 3, 4.

Метод трассировки лучей

Главная идея этого алгоритма была предложена в 196 г. А. Аппелем, а первая реализация была выполнена в 1971 г.

Наблюдатель видит любой объект посредством испускаемого неким источником света, который падает на этот объект, отражается или преломляется согласно законам оптики и затем каким-то путем доходит до глаза наблюдателя. Из огромного множества лучей света, выпущенных источником, лишь небольшая часть дойдет до наблюдателя. Следовательно, отслеживать пути лучей в таком порядке неэффективно с точки зрения вычислений. Аппель предложил отслеживать (трассировать) лучи в обратном направлении, т. е. от наблюдателя к объекту. В первой реализации этого метода трассировка прекращалась, как только луч пересекал поверхность видимого непрозрачного объекта; т. е. луч использовался только для обработки скрытых или видимых поверхностей. Впоследствии были реализованы алгоритмы трассировки лучей с использованием более полных моделей освещения с учетом эффектов отражения одного объекта от поверхности другого, преломления, прозрачности и затенения.

В этом алгоритме предполагается, что сцена уже преобразована в про-

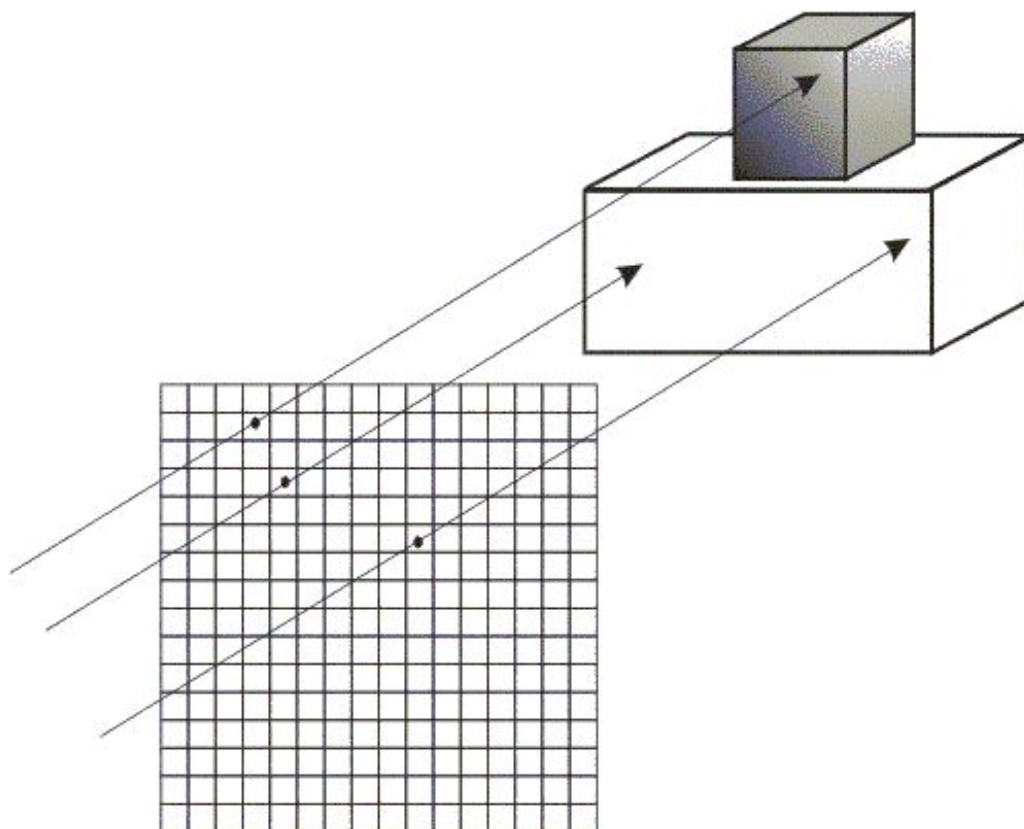


Рис. 23: Трассировка параллельными лучами.

странство изображения. Если используется ортографическая проекция, то точка зрения или наблюдатель находится в бесконечности на положительной полуоси OZ . В этом случае все световые лучи, идущие от наблюдателя, параллельны оси (рис. 23). Каждый луч проходит через пиксель раstra до сцены. Траектория каждого луча отслеживается, чтобы определить, какие именно объекты сцены, если таковые существуют, пересекаются с данным лучом. Необходимо проверить пересечение каждого объекта сцены с каждым лучом. Если луч пересекает объект, то определяются все возможные точки пересечения луча и объекта. Можно получить большое количество пересечений, если рассматривать много объектов. Эти пересечения упорядочиваются по глубине. Пересечение с максимальным значением представляет видимую поверхность для данного пикселя. Атрибуты этого объекта используются для определения характеристик пикселя.

Если точка зрения находится не в бесконечности (перспективная проек-

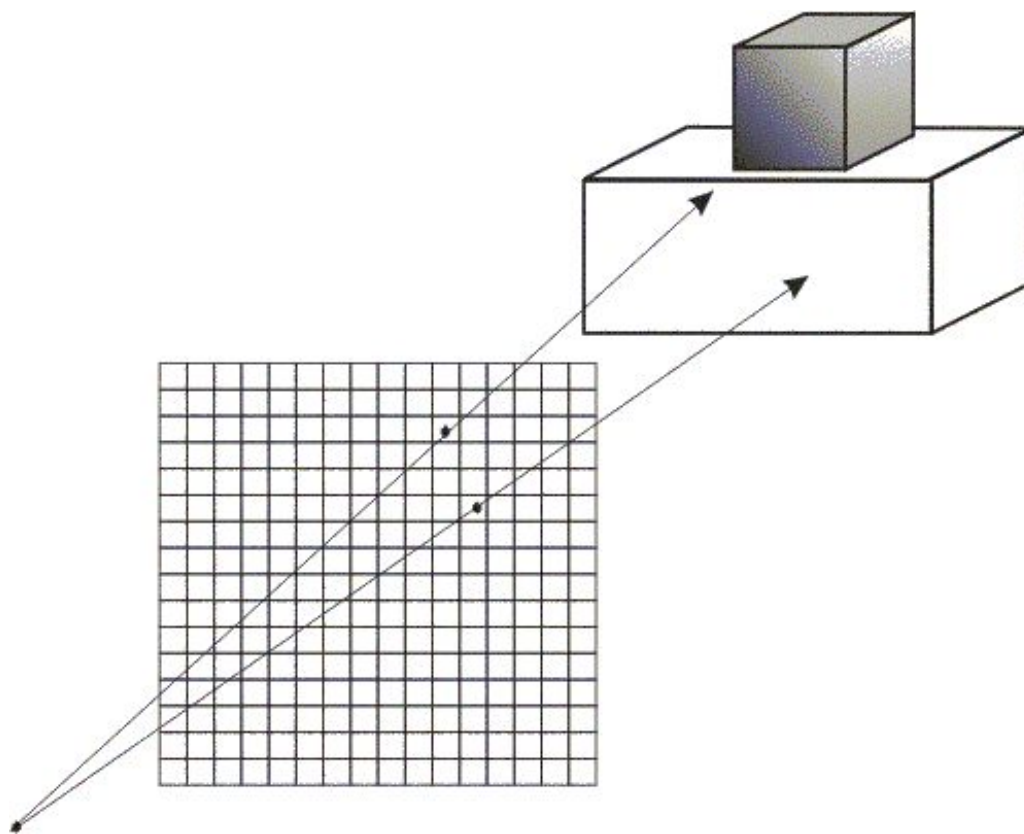


Рис. 24: Трассировка с центральной точкой.

ция), алгоритм трассировки лучей лишь незначительно усложняется. Здесь предполагается, что наблюдатель по-прежнему находится на положительной полуоси OZ . Картинная плоскость, т. е. растр, перпендикулярна оси OZ , как показано на рис. 24.

Наиболее важным и трудоемким элементом этого алгоритма является процедура определения пересечений, поскольку эта задача отнимает наибольшую часть времени всей работы алгоритма. Поэтому эффективность методов поиска особенно важна. Объекты сцены могут состоять из набора плоских многоугольников, многогранников или тел, ограниченных замкнутыми параметрическими поверхностями. Для ускорения поиска важно иметь эффективные критерии, позволяющие исключить из процесса заведомо лишние объекты.

Одним из способов сокращения числа пересекаемых объектов является погружение объектов в выпуклую оболочку — сферическую или в форме парал-

лелепипеда. Поиск пересечения с такой оболочкой очень прост, и если луч не пересекает оболочки, то не нужно больше искать пересечений этого объекта с лучом.

Особенно просто выполняется тест на пересечение со сферической оболочкой. Несколько большего объема вычислений требует задача о пересечении с прямоугольным параллелепипедом, поскольку необходимо проверить пересечение луча по меньшей мере с тремя бесконечными плоскостями, ограничивающими прямоугольную оболочку. Поскольку точки пересечения могут оказаться вне граней этого параллелепипеда, для каждой из них следует, кроме того, произвести проверку на попадание внутрь. Следовательно, для трех измерений тест с прямоугольной оболочкой оказывается более медленным, чем тест со сферической оболочкой.

После выполнения этих первичных тестов начинается процесс поиска пересечений с объектами, попавшими в список потенциально видимых. При этом задача формирования изображения не исчерпывается нахождением самой точки пересечения: если мы учитываем эффекты отражения и преломления, необходимо отслеживать дальнейший путь луча, для чего, как правило, требуется восстановить нормаль к поверхности, а также определить направление отраженного или преломленного луча. В связи со всеми этими задачами важно выбрать достаточно удобные аппроксимации поверхностей, составляющих сцену. Определение атрибутов пикселя, выводимого в конечном итоге на экран, зависит от выбора модели освещения.

Алгоритм трассировки лучей для простых непрозрачных поверхностей можно представить следующим образом.

Создать список объектов, содержащий:

- полное описание объекта: тип, поверхность, характеристики, тип оболочки и т. п.;
- описание оболочки: центр и радиус для сферы или шесть значений для параллелепипеда $(x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$.

Для каждого трассируемого луча:

Выполнить для каждого объекта трехмерный тест на пересечение с оболочкой. Если луч пересекает эту оболочку, то занести объект в список активных объектов.

Если список активных объектов пуст, то изобразить данный пиксель с фоновым значением цвета и продолжать работу. В противном случае для каждого объекта из списка активных объектов:

- Найти пересечения со всеми активными объектами.
- Если список пересечений пуст, то изобразить данный пиксель с фоновым значением цвета.
- В противном случае в списке пересечений найти ближайшее к наблюдателю (с максимальным значением z) и определить атрибуты точки.
- Изобразить данный пиксель, используя найденные атрибуты пересеченного объекта и соответствующую модель освещенности.

В настоящее время алгоритм трассировки, несмотря на вычислительную сложность, стал очень популярен, особенно в тех случаях, когда время формирования изображения не очень существенно, но хочется добиться как можно большей реалистичности изображения.

Задание

По указанию преподавателя реализовать не менее трех алгоритмов удаления невидимых линий и поверхностей.

Контрольные вопросы

1. В чем заключается суть удаления невидимых линий и поверхностей?
2. В каком пространстве работает алгоритм Робертса?
3. Для каких объектов примеряется алгоритм Робертса?

4. Что представляет собой вектор-столбец обобщенной матрицы описания многогранника?
5. Как интерпретируется выражение $(\vec{r} \cdot M) \geq 0$ (M — обобщенная матрица) в алгоритме Робертса?
6. В каком пространстве работает алгоритм Варнока?
7. Какие типы расположения многоугольника относительно окна рассматриваются в алгоритме Варнока?
8. Который из шести шагов алгоритма решает задачу об удалении невидимых поверхностей?
9. В каком пространстве работает алгоритм Вейлера-Азертонна?
10. В чем принципиальное отличие алгоритма Вейлера-Азертонна от алгоритма Варнока?
11. Какое обобщение алгоритма Вейлера-Азертонна предложил Кэтмул?
12. Кем предложен алгоритм Z-буфера?
13. В чем недостатки алгоритма Z-буфера?
14. На чем основаны методы приоритетов?
15. Для какого вида изображения разработан метод художника?
16. Для какого вида изображения разработан метод плавающего горизонта?
17. Что общего между алгоритмом построчного сканирования и методом Z-буфера?
18. В чем состоит идея метода трассировки?
19. Какие бывают виды трассировки?

20. Какие приемы используются для повышения эффективности алгоритма трассировки?

Литература

1. Шикин, Е. В., Боресков, А. В. Компьютерная графика. Полигональные модели [Текст]. — М.: ДИАЛОГ-МИФИ, 2001. — 464 с.
2. Роджерс, Д. Алгоритмические основы машинной графики [Текст]. — М.: Мир, 1989. — 512 с.
3. Куликов, А. И., Овчинникова, Т. Э. Алгоритмические основы современной компьютерной графики [Электронный ресурс]. — Интернет университет информационных технологий intuit.ru, 2007. — Режим доступа: <http://www.intuit.ru/department/graphics/graphalg/>. — Загл. с экрана.

9 Алгоритмы закрашки

Цель работы

Целью выполнения данной работы является изучение основных алгоритмов закрашки и получение навыков по их реализации и использованию.

Теоретические сведения

Визуальное восприятие объектов окружающей действительности представляет собой сложный процесс, имеющий как физические, так и психологические аспекты.

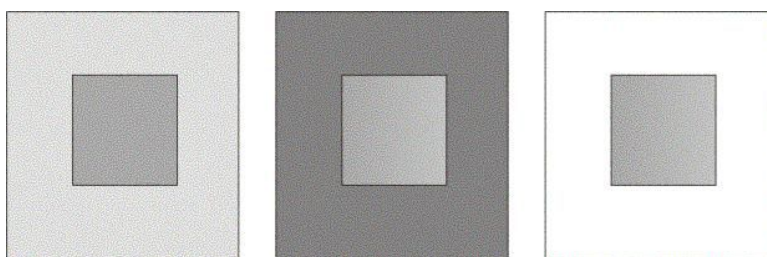


Рис. 25: Эффекты восприятия изображения.

Глаз адаптируется к средней яркости рассматриваемой сцены, поэтому при смене фона изменяется восприятие сцены. Например, однородно окрашенная область на более темном фоне будет казаться более яркой, чем на светлом. Кроме того, она будет восприниматься как более обширная (рис. 25).

Еще одна особенность восприятия заключается в том, что граница равномерно освещенной области кажется более яркой по сравнению с внутренними частями. Это явление было обнаружено Эрнстом Махом, поэтому оно получило название **эффекта полос Маха**. Такие особенности необходимо учитывать, если мы стремимся к созданию реалистических изображений сцен.

При формировании изображения сцен, содержащих зеркальные и полупрозрачные поверхности, следует использовать законы геометрической оптики, преломляющие свойства материалов, эффекты смешения цветов и т. д.

Простая модель освещения

Объекты окружающего пространства становятся видимыми для глаза благодаря световой энергии, которая может излучаться поверхностью предмета, отражаться или проходить сквозь нее. В свою очередь, отражение света от поверхности зависит от физических свойств материала, из которого она изготовлена, а также от характера и расположения источника света. Яркость (или интенсивность) освещения зависит от энергии светового потока, которая обуславливается, во-первых, мощностью источника света, а во-вторых, отражающими и пропускающими свойствами объекта.

Мы рассмотрим модель освещения, учитывающую только отражение. Свойства отраженного света зависят главным образом от направления лучей и характеристик отражающей поверхности.

Отражение может быть двух видов: **диффузное** и **зеркальное**. Первое из них возникает в ситуации, когда свет как бы проникает под поверхность объекта, поглощается, а потом равномерно излучается во всех направлениях. Поверхность в этом случае рассматривается как идеальный рассеиватель. При этом возникает эффект матового света, а видимая освещенность того или иного участка поверхности не зависит от положения наблюдателя. Зеркальное отражение, наоборот, происходит от внешней поверхности, интенсивность его неоднородна, поэтому видимый максимум освещенности зависит от положения глаза наблюдателя.

Свет точечного источника отражается от поверхности рассеивателя по закону Ламберта: интенсивность отражения пропорциональна косинусу угла между внешней нормалью к поверхности и направлением к источнику света (рис. 26). Если I_S — интенсивность источника света, φ — угол между вектором внешней нормали к поверхности и направлением к источнику света, то интенсивность отраженного света определяется формулой

$$I = \begin{cases} I_S \cos \varphi & \text{при } 0 \leq \varphi \leq \pi/2, \\ 0 & \text{в противном случае.} \end{cases} \quad (10)$$

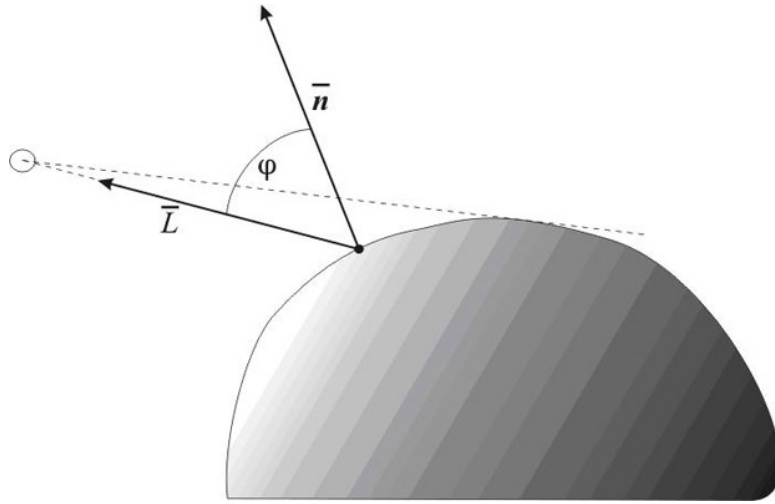


Рис. 26: Освещение точечным источником.

При таком расчете интенсивности получится очень контрастная картина, т. к. участки поверхности, на которые лучи от источника не попадают напрямую, останутся абсолютно черными. Для повышения реалистичности необходимо учитывать рассеивание света в окружающем пространстве. Поэтому вводится **фоновая освещенность**, зависящая от интенсивности рассеянного света I_F , и интенсивность отраженного света определяется выражением

$$I = \begin{cases} I_F k_F + k_s I_S \cos \varphi & \text{при } 0 \leq \varphi \leq \pi/2, \\ I_F k_F & \text{в противном случае,} \end{cases} \quad (11)$$

где k_F — коэффициент диффузного отражения рассеянного света, k_S — коэффициент диффузного отражения падающего света, $0 \leq k_S \leq 1$, $0 \leq k_F \leq 1$.

В описанной модели пока никак не учитывалась удаленность источника света от поверхности, поэтому по освещенности двух объектов нельзя судить об их взаимном расположении в пространстве. Если мы хотим получить перспективное изображение, то необходимо включить затухание интенсивности с расстоянием. Обычно интенсивность света обратно пропорциональна квадрату расстояния от источника. В качестве расстояния до источника в случае перспективного преобразования можно взять расстояние до центра проекции, и если он достаточно удален, то изображение будет достаточно адекватным.

Но если этот центр расположен близко к объекту, то квадрат расстояния меняется очень быстро, и в этом случае лучше использовать линейное затухание. В этом случае интенсивность отраженного света от непосредственно освещенных участков поверхности будет задаваться формулой

$$I = I_F k_F + \frac{k_S I_S \cos \varphi}{d + C}, \quad (12)$$

где d — расстояние до центра проекции, а C — произвольная постоянная. Если центр проекции находится на бесконечности, т. е. при параллельном проецировании, то в качестве d можно взять расстояние до объекта, наиболее близкого к наблюдателю.

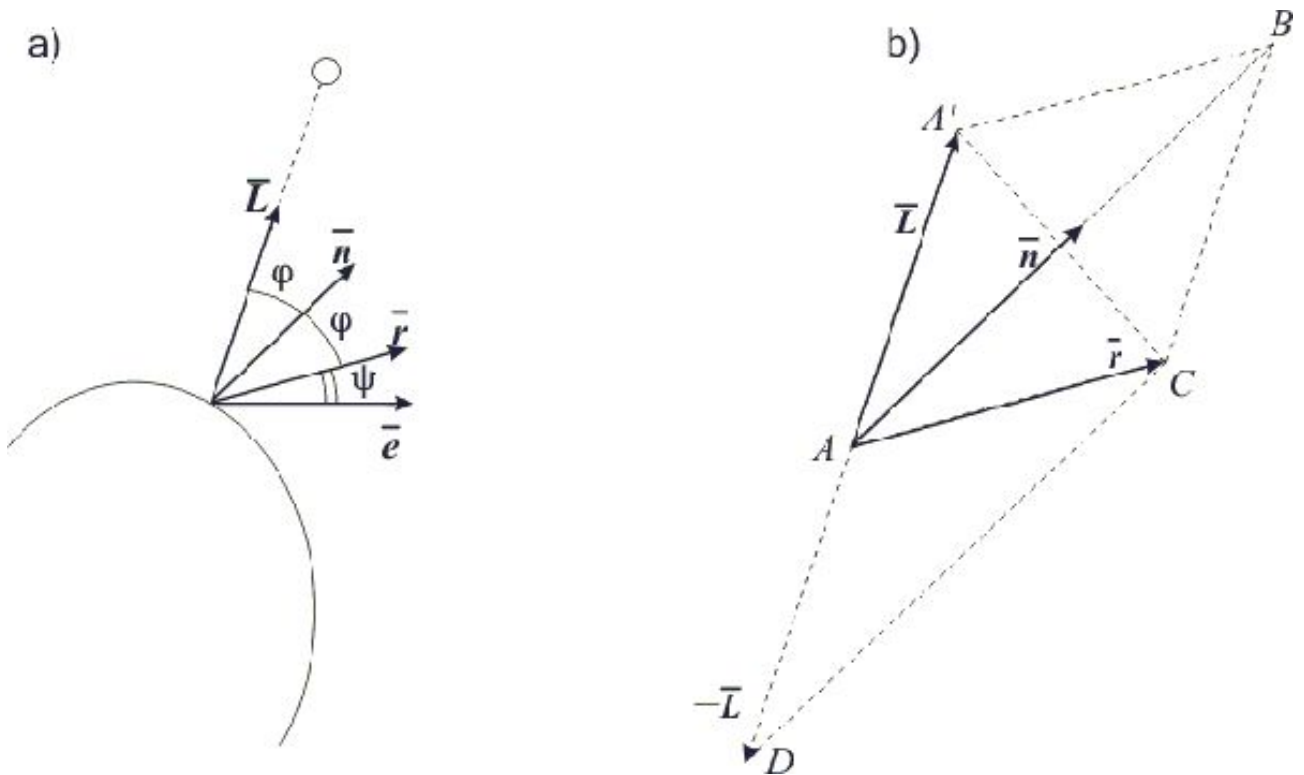


Рис. 27: Зеркальное отражение.

В отличие от диффузного, зеркальное отражение является направленным. Идеальное зеркало отражает лучи по принципу «отраженный и падающий лучи лежат в одной плоскости, причем угол падения равен углу отражения» (имеется в виду угол между направлением луча и нормалью к поверхности). Если поверхность не идеально зеркальная, то лучи отражаются в различных

направлениях, но с разной интенсивностью, а функция изменения интенсивности имеет четко выраженный максимум. Поскольку физические свойства зеркального отражения довольно сложны, то в компьютерной графике используется эмпирическая модель Фонга. Суть ее заключается в том, что для глаза наблюдателя интенсивность зеркально отраженного луча зависит от угла между идеально отраженным лучом и направлением к наблюдателю (рис. 27 а). Кроме того, поскольку зеркальное отражение зависит еще и от длины волны, это также будем учитывать в формуле для вычисления интенсивности. Модель Фонга описывается соотношением

$$I_Z = \omega(\varphi, \lambda) \cdot I_S \cos^n \psi, \quad (13)$$

где $\omega(\varphi, \lambda)$ — функция отражения, λ — длина волны. Степень, в которую возводится косинус угла, влияет на размеры светового блика, наблюдаемого зрителем. Графики этой функции приведены на рис. 28, и они как раз являются характерными кривыми поведения функции изменения интенсивности в зависимости от свойств поверхности.

Теперь модель освещенности, учитывающую зеркальное и диффузное отражения, можно описать формулой

$$I = I_F k_F + \frac{I_S (k_s \cos \varphi + \omega(\varphi, \lambda) \cdot \cos^n \psi)}{d + C}. \quad (14)$$

Используя единичные векторы \vec{L} (направление к источнику) и \vec{n} (внешняя нормаль), косинус угла φ можно вычислить через скалярное произведение: $\cos \varphi = (\vec{L} \cdot \vec{n})$. Для расчета интенсивности зеркального отражения сначала надо определить отраженный вектор \vec{r} . Из рис. 27 б видно, что $\vec{r} = |\overline{AB}| \cdot \vec{n} - \vec{L}$. С другой стороны \overline{AB} является диагональю ромба $AA'BC$, поэтому $|\overline{AB}| = 2 \cdot (\vec{L} \cdot \vec{n})$. Учитывая все эти соотношения, полу-

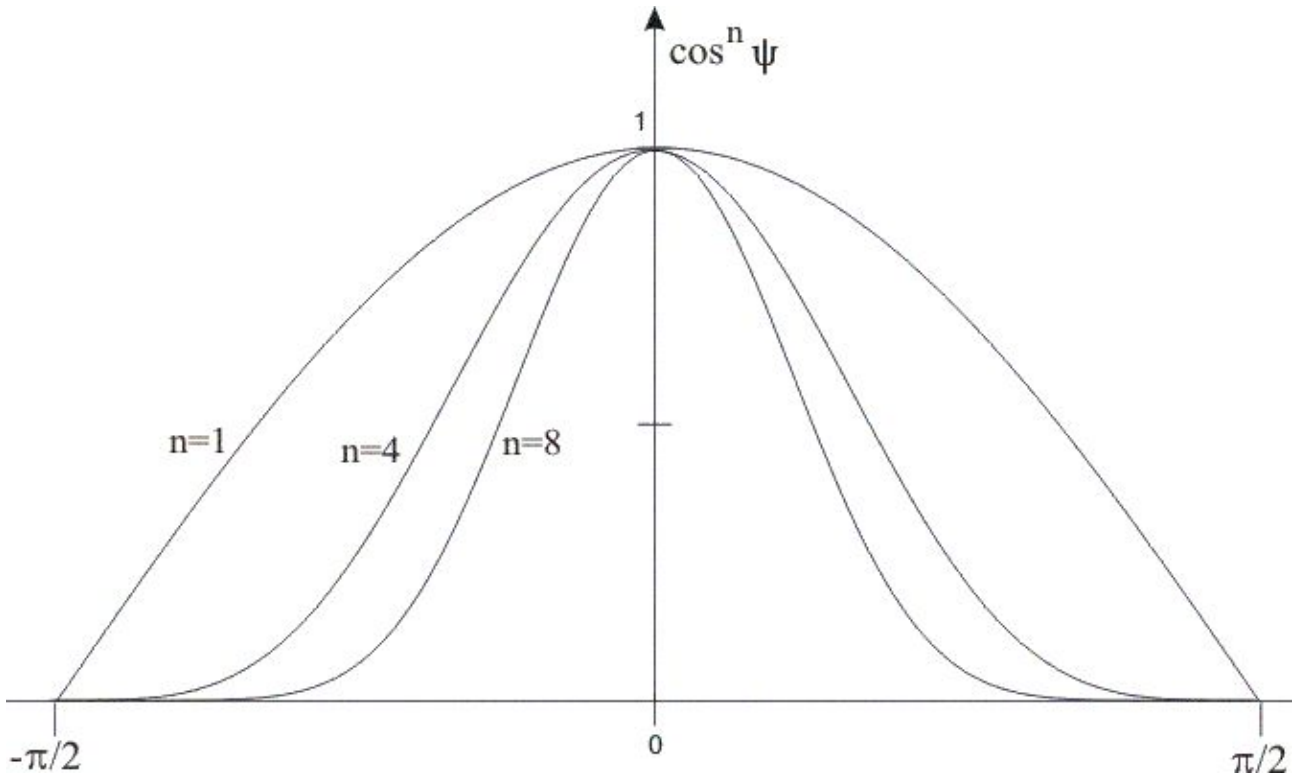


Рис. 28: Функция изменения интенсивности.

чаем формулу

$$I = I_F k_F + \frac{I_S \{k_s(\vec{L} \cdot \vec{n}) + \omega(\varphi, \lambda) \cdot [2 \cdot (\vec{L} \cdot \vec{n}) \cdot (\vec{e} \cdot \vec{n}) - (\vec{e} \cdot \vec{L})]^n\}}{d + C}. \quad (15)$$

В алгоритмах закрашивания с использованием цветовых моделей интенсивность рассчитывается для каждого из базовых цветов, поскольку изменение интенсивности при зеркальном отражении зависит от длины волны.

Закраска граней

Плоское закрашивание. Если предположить, что источник света находится на бесконечности, то лучи света, падающие на поверхность, параллельны между собой. Если к этому добавить условие, что наблюдатель находится в бесконечно удаленной точке, то эффектом ослабления света с увеличением расстояния от источника также можно пренебречь. Кроме того, такое положение наблюдателя означает еще и то, что векторы, направленные от разных

точек поверхности к наблюдателю, также будут параллельны. При выполнении всех этих условий, как следует из формулы (15), плоская грань во всех точках имеет одинаковую интенсивность освещения, поэтому она закрашивается одним цветом. Такое закрашивание называется **плоским**.

Если мы аппроксимируем некоторую гладкую поверхность многогранником, то при плоском закрашивании неизбежно проявятся ребра, поскольку соседние грани с различными направлениями нормалей имеют разный цвет. Эффект полос Маха дополнительно усиливает этот недостаток. Для его устранения при использовании этого способа закрашивания можно лишь увеличить число граней многогранника, что приводит к увеличению вычислительной сложности алгоритма.

Закраска методом Гуро. Один из способов устранения дискретности интенсивностей закрашивания был предложен Гуро. Его метод заключается в том, что используются не нормали к плоским граням, а нормали к аппроксимируемой поверхности, построенные в вершинах многогранника. После этого вычисляются интенсивности в вершинах, а затем во всех внутренних точках многоугольника выполняется билинейная интерполяция интенсивности.

Метод сочетается с алгоритмом построения сканирования. После того как грань отображена на плоскость изображения, для каждой сканирующей строки определяются ее точки пересечения с ребрами. В этих точках интенсивность вычисляется с помощью линейной интерполяции интенсивностей в вершинах ребра. Затем для всех внутренних точек многоугольника, лежащих на сканирующей строке, также вычисляется интенсивность методом линейной интерполяции двух полученных значений. На рис. 29 показан плоский многоугольник с вычисленными значениями интенсивностей в вершинах.

Пусть I_A, I_B, I_C — интенсивности в вершинах A, B, C , x_A, x_B, x_C — горизонтальные координаты этих точек. Тогда в точках пересечения сканирующей строки с ребрами многоугольника интенсивности можно вычислить по

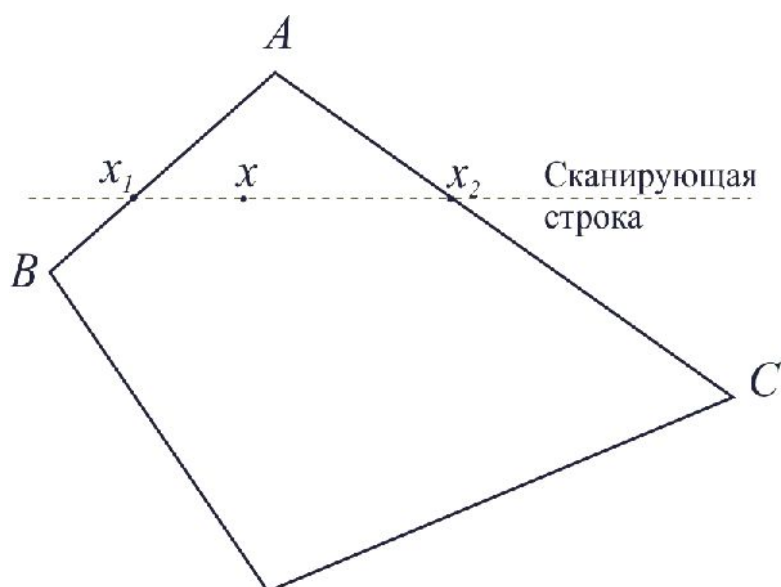


Рис. 29: Интерполяция интенсивности.

формулам интерполяции:

$$I_1 = t_1 I_A + (1 - t_1) I_B, \quad t_1 = \frac{x_1 - x_B}{x_A - x_B}, \quad I_2 = t_2 I_A + (1 - t_2) I_C, \quad t_2 = \frac{x_2 - x_C}{x_A - x_C}. \quad (16)$$

После этого интенсивность в точке x получаем путем интерполяции значений на концах отрезка:

$$I = t I_1 + (1 - t) I_2, \quad t = \frac{x_2 - x}{x_2 - x_1}. \quad (17)$$

К недостаткам метода Гуро следует отнести то, что он хорошо работает только с диффузной моделью отражения. Форма бликов на поверхности и их расположение не могут быть адекватно воспроизведены при интерполяции на многоугольниках. Кроме того, есть проблема построения нормалей к поверхности. В алгоритме Гуро нормаль в вершине многогранника вычисляется путем усреднения нормалей к граням, примыкающим к этой вершине. Такое построение сильно зависит от характера разбиения.

Закраска методом Фонга. Фонг предложил вместо интерполяции интенсивностей произвести интерполяцию вектора нормали к поверхности на

сканирующей строке. Этот метод требует больших вычислительных затрат, поскольку формулы интерполяции (15)–(16) применяются к трем компонентам вектора нормали, но зато дает лучшую аппроксимацию кривизны поверхности. Поэтому зеркальные свойства поверхности воспроизводятся гораздо лучше.

Нормали к поверхности в вершинах многогранника вычисляются так же, как и в методе Гуро. А затем выполняется билинейная интерполяция в сочетании с построчным сканированием. После построения вектора нормали в очередной точке вычисляется интенсивность.

Этот метод позволяет устранить ряд недостатков метода Гуро, но не все. В частности, эффект полос Маха в отдельных случаях в методе Фонга бывает даже сильнее, хотя в подавляющем большинстве случаев аппроксимация Фонга дает лучшие результаты.

Задание

По указанию преподавателя реализовать не менее двух алгоритмов закрашивания.

Контрольные вопросы

1. Что такое эффект полос Маха?
2. Чем отличается диффузное отражение от зеркального?
3. От чего зависит интенсивность освещения точки поверхности при диффужном отражении?
4. От чего зависит интенсивность освещения точки поверхности при зеркальном отражении?
5. Какие параметры учитывает модель зеркального отражения, предложенная Фонгом?

6. Меняется ли интенсивность освещения при плоском закрашивании грани?
7. Какой параметр интерполируется при закрашивании методом Гуро?
8. Какой параметр интерполируется при закрашивании методом Фонга?

Литература

1. Шикин, Е. В., Боресков, А. В. Компьютерная графика. Полигональные модели [Текст]. — М.: ДИАЛОГ-МИФИ, 2001. — 464 с.
2. Роджерс, Д. Алгоритмические основы машинной графики [Текст]. — М.: Мир, 1989. — 512 с.
3. Куликов, А. И., Овчинникова, Т. Э. Алгоритмические основы современной компьютерной графики [Электронный ресурс]. — Интернет университет информационных технологий intuit.ru, 2007. — Режим доступа: <http://www.intuit.ru/department/graphics/graphalg/>. — Загл. с экрана.

Литература

1. GNU Image Manipulation Program. Руководство пользователя [Электронный ресурс]. — GNU Image Manipulation Program, 2010. — Режим доступа: <http://docs.gimp.org/2.6/ru/>. — Загл. с экрана.
2. Inkscape tutorial [Электронный ресурс]. — Inkscape.org, 2009. — Режим доступа: <http://inkscape.org/doc/basic/tutorial-basic.ru.html>. — Загл. с экрана.
3. Google SketchUp [Электронный ресурс]. — Google SketchUp, 2010. — Режим доступа: <http://sketchup.google.com/index.html>. — Загл. с экрана.
4. Шикин, Е. В., Боресков, А. В. Компьютерная графика. Полигональные модели [Текст]. — М.: ДИАЛОГ-МИФИ, 2001. — 464 с.
5. Семёнов, А. Б. Обработка и анализ изображений с использованием языка JAVA [Электронный ресурс]. — Sun Microsystems — Россия и СНГ — Материалы курсов, 2007. — Режим доступа: http://ru.sun.com/research/materials/Semenov_Java_Seminar.jsp. — Загл. с экрана.
6. Роджерс, Д., Адаме, Дж. Математические основы машинной графики [Текст]. — М.: Мир, 2001. — 604 с.
7. Роджерс, Д. Алгоритмические основы машинной графики [Текст]. — М.: Мир, 1989. — 512 с.
8. Никулин, Е. А. Компьютерная геометрия и алгоритмы машинной графики [Текст]. — СПб.: БХВ-Петербург, 2003. — 560 с.
9. Порев В. Н. Компьютерная графика [Текст]. — СПб.: БХВ-Петербург, 2002. — 432 с.
10. Тихомиров Ю. Программирование трехмерной графики [Текст]. — СПб.: БХВ-Санкт-Петербург, 1998. — 256 с.

11. Эйнджел, Э. Интерактивная компьютерная графика. Вводный курс на базе OpenGL [Текст]. — М.: Издательский дом «Вильямс», 2001. — 592 с.
12. Хилл, Ф. Программирование компьютерной графики. Для профессионалов [Текст]. — СПб.: Питер, 2002. — 1088 с.
13. Куликов, А. И., Овчинникова, Т. Э. Алгоритмические основы современной компьютерной графики [Электронный ресурс]. — Интернет университет информационных технологий intuit.ru, 2007. — Режим доступа: <http://www.intuit.ru/department/graphics/graphalg/>. — Загл. с экрана.
14. Павлидис, Т. Алгоритмы машинной графики и обработки изображений [Текст]. — М.: Радио и связь, 1986. — 400 с.
15. Петров, М. Н., Молочков, В. П. Компьютерная графика [Текст]. — СПб.: Питер, 2003. — 736 с.