ГЛАВА 6

Двухмерное наблюдение

Сад колибри, нарисованный Джоном Дерри (John Derry) из Time Arts, Inc.,
с использованием графического планшета и беспроводного самопишущего пера
(перепечатано с разрешения Wacom Technology Corporation)

В главе 2 кратко рассматривались концепции и функции двухмерного наблюдения. В данной главе мы более подробно опишем процедуры вывода на экран проекции двухмерного изображения. Обычно графический пакет позволяет пользователю задавать, какую часть определенной картины нужно отобразить на экран, и где эта часть должна располагаться на дисплее. Чтобы определить изображение, можно использовать любую удобную декартову систему координат, называемую *внешней*. Для двухмерного изображения проекция выбирается следующим образом: задается область плоскости xy, которая содержит все изображение или его часть. Пользователь может выбирать одну область для вывода на экран, кроме того, может одновременно изображаться несколько областей или анимированные панорамы сцены. Затем части изображения в выбранных областях отображаются в области, заданные в координатах устройства. Если выбрано несколько областей наблюдения, их можно поместить в различных местах дисплея либо ввести фрагменты в бо́льшие участки. Двухмерное преобразование наблюдения из внешних координат в координаты устройства включает трансляцию, вращение и масштабирование, а также удаление тех частей изображения, которые лежат вне границ выбранной области сцены.

6.1. ДВУХМЕРНЫЙ КОНВЕЙЕР НАБЛЮДЕНИЯ

Участок двухмерной сцены, выбранный для отображения на экране, называется *отсекающим окном*, поскольку все части сцены вне выбранного участка, “отсекаются”, и на экран выводится только часть сцены, которая находится внутри отсекающего окна. Иногда отсекающее окно называется *окном наблюдения* или *смотровым окном*.

Рис. 6.1. Экран с несколькими окнами (*перепечатано с разрешения Sun Microsystems*)

Одно время в графических системах отсекающее окно называлось просто “окном”, но сейчас в мире компьютеров столько различных окон, что нужно как-то их различать. Например, система управления окнами может для вывода на экран графики и текста создавать несколько областей на дисплее, каждая из которых называется “окном” (рис. 6.1). Поэтому далее мы всегда будем использовать термин *отсекающее окно* для обозначения участка сцены, который в конечном итоге будет преобразован в пиксельную структуру в окне изображения на экране дисплея. Отметим, что графические пакеты также позволяют контролировать расположение окна изображения на экране дисплея с помощью другого “окна”, называемого *полем просмотра* (viewport). Сначала объекты сцены, находящиеся внутри отсекающего окна, отображаются в поле просмотра, которое затем размещается в окне на экране дисплея. Отсекающее окно задает, *что* мы хотим увидеть, а поле просмотра указывает, *где* это будет располагаться на устройстве вывода.

Меняя положение поля просмотра, можно представлять объект в различных местах экрана устройства вывода. Для отображения различных участков сцены в различных точках экрана можно использовать несколько полей просмотра. Кроме того, меняя размер полей просмотра, можно менять размер и пропорции отображенных объектов. Для достижения эффектов масштабирования отсекающие окна разного размера последовательно отображаются в поле просмотра фиксированной величины. При уменьшении отсекающего окна мы увеличиваем некоторую часть сцены, и становятся видны детали, незаметные при б*о*льших отсекающих окнах. Таким образом, б*о*льший обзор достигается за счет уменьшения масштаба участка сцены с помощью последовательно увеличивающихся отсекающих окон. Наконец, чтобы получить панорамный эффект, нужно перемещать отсекающее окно фиксированного размера вдоль различных объектов сцены.

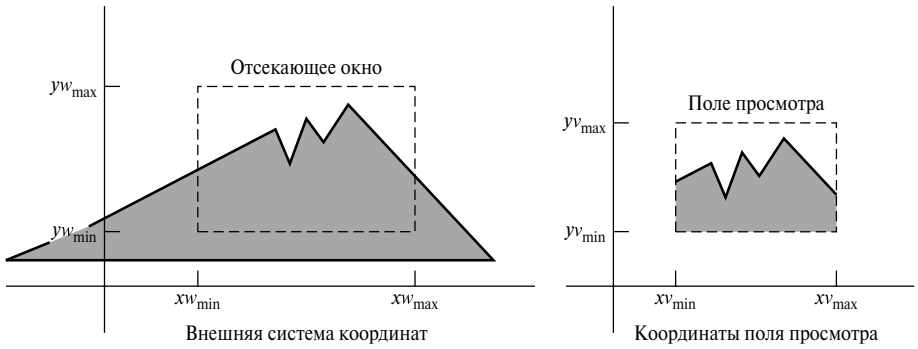


Рис. 6.2. Отсекающее окно и соответствующее поле просмотра, заданные в виде треугольников, ориентированных по координатным осям

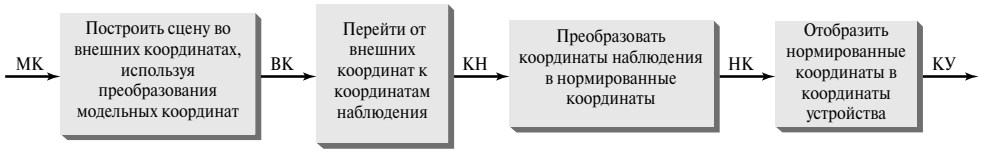


Рис. 6.3. Двухмерный конвейер преобразований наблюдения

Обычно отсекающие окна и поля просмотра — это прямоугольники стандартной ориентации, в которых стороны параллельны координатным осям. В некоторых приложениях используются иные геометрии окон или полей, например, многоугольные формы и окружности, но обработка таких форм занимает больше времени. Рассмотрим вначале только прямоугольные поля просмотра и отсекающие окна, подобные показанным на рис. 6.2.

Отображение двухмерного описания сцены во внешних координатах в координаты устройства называется *двухмерным преобразованием точки наблюдения*. Иногда это преобразование называется просто *преобразованием окна в поле просмотра* или *преобразованием окна*. Однако, в общем случае, наблюдение — это не только преобразование из координат отсекающего окна в координаты поля просмотра. По аналогии с трехмерным наблюдением этапы двухмерного наблюдения можно описать так, как показано на рис. 6.3. После построения сцены во внешних координатах можно задать отдельную двухмерную *систему отсчета в координатах наблюдения* (систему наблюдения), в которой определяется отсекающее окно. Впрочем, отсекающее окно часто определяется просто во внешних координатах, так что координаты наблюдения в двухмерных приложениях не отличаются от внешних координат. (Для трехмерной сцены, впрочем, нужна дополнительная система наблюдения, в которой указываются параметры положения и ориентации наблюдателя и направление наблюдения.)

Чтобы процесс наблюдения был независимым от требований устройств вывода, в графических системах описания объектов переводятся в нормированные координаты, и применяются процедуры отсечения. В одних системах используются нормированные координаты с диапазоном от 0 до 1, в других — с диапазоном от -1 до 1. В зависимости от используемой графической библиотеки поле просмотра определяется либо в нормированных, либо в экранных координатах после нормировки. На последнем этапе преобразования наблюдения содержимое поля просмотра передается в точки окна на экране дисплея.

Отсечение обычно выполняется в нормированных координатах. Это позволяет сократить вычисления, объединив вначале различные матрицы преобразования. Отметим, что процедуры отсечения имеют первостепенную важность в компьютерной графике. Они используются не только в преобразованиях наблюдения, но и в системах управления окнами, пакетах рисования для стирания участков картины, а также во множестве других приложений.

6.2. ОТСЕКАЮЩЕЕ ОКНО

Чтобы добиться определенного эффекта наблюдения в программе-приложении, можно разработать свое отсекающее окно с произвольными выбранными формой, размером и ориентацией. Например, в качестве отсекающего окна можно выбрать звездообразную форму или эллипс, а также фигуру со сплайновыми границами. В то же время, чтобы обрезать сцену с использованием вогнутого многоугольника или отсекающего окна с нелинейными границами, необходимо больше обработки, чем для обрезки с использованием прямоугольника. Чтобы определить, где объект пересекает окружность, нужно больше вычислений, чем при определении точки пересечения с прямой линией. Отметим также, что в простейшем случае края окна отсечения параллельны координатным осям. Поэтому графические пакеты в общем случае разрешают использовать только прямоугольное отсекающее окно, стороны которого идут по осям x и y.

Если требуется, чтобы отсекающее окно имело какую-то другую форму, можно реализовать свои алгоритмы отсечения и преобразования координат. Кроме того, можно просто отредактировать изображение и получить определенную форму кадра изображения, вмещающего сцену. Например, можно обрезать края картины с помощью любого удобного шаблона, наложив многоугольники, закрашенные цветом фона. Таким образом можно генерировать любые граничные эффекты или даже создавать на изображении внутренние дыры.

Прямоугольные отсекающие окна стандартной ориентации легко определяются координатами двух противоположных углов. Если требуется получить повернутое изображение сцены, можно либо определить прямоугольное отсекающее окно в повернутой системе координат изображения, либо (что равнозначно) повернуть сцену во внешних координатах. Некоторые системы позволяют поворачивать двухмерный кадр наблюдения, но обычно отсекающее окно задается во внешних координатах.

Рис. 6.4. Повернутое отсекающее окно, определенное в координатах наблюдения

ОТСЕКАЮЩЕЕ ОКНО В КООРДИНАТАХ НАБЛЮДЕНИЯ

Обычно при двухмерном преобразовании наблюдения во внешней системе координат задается *система наблюдения*. Данная система отсчета позволяет задавать прямоугольное отсекающее окно с любой выбранной ориентацией и положением, как показано на рис. 6.4. Чтобы получить проекцию сцены во внешних координатах, определяемую отсекающим окном, изображенным на рис. 6.4, нужно просто перевести описание сцены в координаты наблюдения. Хотя многие графические пакеты не имеют функций задания отсекающего окна в двухмерной системе координат наблюдения, при определении отсекаемой области трехмерной сцены данный подход является стандартным.

Начало двухмерной системы координат наблюдения выбирается в некоторой точке внешней системы координат $\mathbf{P}_0 = (x_0, y_0)$, а ориентацию можно задать, используя внешний вектор $\mathbf{V}$, который определяет направление y_{view}. Вектор $\mathbf{V}$ называется двухмерным *вектором верха* (view up vector). Альтернативный метод задания ориентации системы координат наблюдения — указать угол поворота относительно оси x или y во внешней системе координат. По этому углу затем можно получить вектор верха. После установки параметров, определяющих систему координат наблюдения, описание сцены переводится в систему наблюдения с использованием процедуры, описанной в разделе 5.8. Это включает ряд преобразований и эквивалентно наложению системы координат наблюдения на внешнюю систему координат.

Первый этап в последовательности преобразований — транслировать начало координат системы наблюдения в начало внешней системы координат. Затем нужно повернуть систему наблюдения, чтобы совместить ее со внешней системой координат. Для данного вектора ориентации $\mathbf{V}$ можно вычислить компоненты единичных векторов $\mathbf{v} = (v_x, v_y)$ и $\mathbf{u} = (u_x, u_y)$ для осей y_{view} и x_{view} соответственно. Данные единичные векторы используются для формирования первой и второй строки матрицы вращения $\mathbf{R}$, переводящей оси системы наблюдения $x_{\text{view}}y_{\text{view}}$ во внешние оси $x_w y_w$.

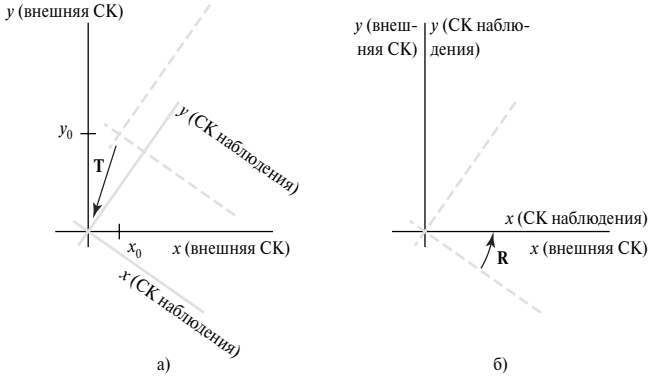


Рис. 6.5. Система наблюдения совмещается с глобальной системой координат за два этапа: а) для перемещения начала системы наблюдения в начало внешней системы координат применяется матрица трансляции $\mathbf{T}$; б) для последующего совмещения осей двух систем используется матрица вращения $\mathbf{R}$

Затем положения объектов во внешних координатах переводятся в координаты наблюдения с помощью сложной матрицы двухмерного преобразования

$$\mathbf{M}_{WC,VC} = \mathbf{R} \cdot \mathbf{T}, \tag{6.1}$$

где $\mathbf{T}$ — матрица трансляции, переносящая начало координат системы наблюдения $\mathbf{P}_0$ в начало внешней системы координат, а $\mathbf{R}$ — матрица вращения, которая поворачивает систему наблюдения до ее совмещения с внешней системой координат. Этапы описанного преобразования координат иллюстрируются на рис. 6.5.

ОТСЕКАЮЩЕЕ ОКНО ВО ВНЕШНИХ КООРДИНАТАХ

Процедура определения стандартного прямоугольного отсекающего окна во внешних координатах обычно предлагается в библиотеке графического программирования. Пользователь просто указывает две точки во внешних координатах — противоположные углы стандартного прямоугольника. После того как отсекающее окно задано, описание сцены обрабатывается с помощью процедур наблюдения и подается на устройство вывода.

Если требуется получить повернутое изображение двухмерной сцены, как обсуждалось в предыдущем разделе, выполняются те же самые действия, но без преобразований, связанных с системой наблюдения. Таким образом, объект простым вращением (и, возможно, трансляцией) переводится в заданное положение, после чего задается отсекающее окно — все это во внешних координатах. В качестве примера рассмотрим повернутое изображение треугольника, изображенное на рис. 6.6, *а* и полученное вращением треугольника в искомое положение и заданием стандартного отсекающего прямоугольника. По аналогии с преобразованием координат, описанным в предыдущем разделе, также можно транслировать треугольник в начало внешней системы координат и определить отсекающее окно вокруг треугольника. В этом слу-

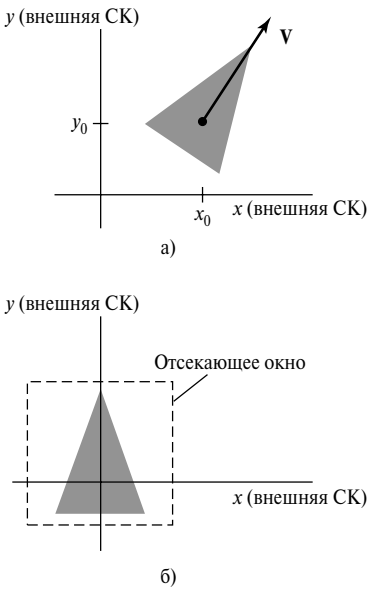


Рис. 6.6. Треугольник (панель *а*) с выбранным началом отсчета и вектором ориентации трансляцией и вращением переводится в отсекающее окно (панель *б*)

чае нужно определить вектор ориентации и выбрать опорную точку, например, центр масс треугольника (приложение А). Затем эта опорная точка транслируется в начало внешней системы координат, и вектор ориентации поворачивается вокруг оси y_{world} с помощью матрицы преобразования (6.1). Имея треугольник искомой ориентации, можно использовать стандартное отсекающее окно во внешних координатах и получить изображение повернутого треугольника. Преобразованное положение треугольника и выбранного отсекающего окна показано на рис. 6.6, *б*.

6.3. НОРМИРОВКА И ПРЕОБРАЗОВАНИЕ ПОЛЯ ПРОСМОТРА

В некоторых графических пакетах нормировка и преобразование окна в поле просмотра объединяются в одну операцию. В этом случае координаты поля просмотра часто задаются в диапазоне от 0 до 1, так что поле просмотра располагается в единичном квадрате. После отсечения единичный квадрат, содержащий поле просмотра, отображается на выходном дисплее. В других системах нормировка и отсечение применяются перед преобразованием поля просмотра. В таких системах границы поля просмотра задаются в экранных координатах относительно точки на дисплее.

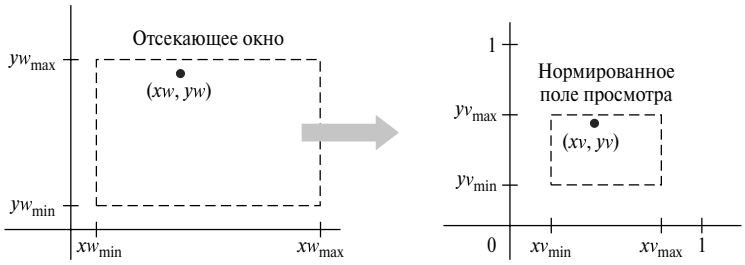


Рис. 6.7. Точка (xw, yw) в отсекающем окне (внешние координаты) отображается в точку (xv, yv) (координаты поля просмотра) в единичном квадрате, так что относительные положения двух точек в соответствующих прямоугольниках не меняются

ОТОБРАЖЕНИЕ ОКНА ОТСЕЧЕНИЯ В НОРМИРОВАННОЕ ПОЛЕ ПРОСМОТРА

Чтобы проиллюстрировать общие процедуры нормировки и преобразования поля просмотра, рассмотрим вначале поле просмотра, определенное нормированными координатами между 0 и 1. Описания объектов переводятся в данное нормированное пространство с помощью преобразования, которое сохраняет то же относительное расположение точек в окне просмотра, которое было в отсекающем окне. Например, если точка находилась в центре отсекающего окна, она отобразится в центр поля просмотра. Данное отображение окна в поле просмотра иллюстрируется на рис. 6.7. Точка (xw, yw) в отсекающем окне переходит в точку (xv, yv) соответствующего поля просмотра.

Чтобы преобразовать точку, заданную во внешних координатах, в соответствующее положение в поле просмотра, нужно положить следующее:

$$\begin{aligned} \frac{xv - xv_{\min}}{xv_{\max} - xv_{\min}} &= \frac{xw - xw_{\min}}{xw_{\max} - xw_{\min}}, \\ \frac{yv - yv_{\min}}{yv_{\max} - yv_{\min}} &= \frac{yw - yw_{\min}}{yw_{\max} - yw_{\min}}. \end{aligned} \tag{6.2}$$

Решая эти уравнения относительно точки в поле просмотра (xv, yv), получаем

$$\begin{aligned} xv &= s_x xw + t_x, \\ yv &= s_y yw + t_y, \end{aligned} \tag{6.3}$$

где масштабные коэффициенты равны:

$$\begin{aligned} s_x &= \frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}}, \\ s_y &= \frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}}, \end{aligned} \tag{6.4}$$

а коэффициенты трансляции —

$$t_x = \frac{xw_{\max} xv_{\min} - xw_{\min} xv_{\max}}{xw_{\max} - xw_{\min}},$$
$$t_y = \frac{yw_{\max} yv_{\min} - yw_{\min} yv_{\max}}{yw_{\max} - yw_{\min}}. \tag{6.5}$$

Поскольку мы просто отображаем точки с внешними координатами в поле просмотра, которое расположено возле начала внешней системы координат, уравнения (6.3) можно также вывести, используя любую последовательность преобразований, переводящих прямоугольник отсекающего окна в прямоугольник поля просмотра. Например, можно получить преобразование внешних координат в координаты поля просмотра, используя следующую последовательность действий.

1. Масштабировать отсекающее окно до размера поля просмотра, используя фиксированную точку ($xw_{\min}$, $yw_{\min}$).
2. Транслировать ($xw_{\min}$, $yw_{\min}$) в ($xv_{\min}$, $yv_{\min}$).

Преобразование масштабирования на этапе 1 можно представить двухмерной матрицей

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & xw_{\min}(1 - s_x) \\ 0 & s_y & yw_{\min}(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix}, \tag{6.6}$$

где s_x и s_y — те же, что и в уравнениях (6.4). Матричное представление трансляции левого нижнего угла отсекающего окна в левый нижний угол поля просмотра выглядит так:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & xv_{\min} - xw_{\min} \\ 0 & 1 & yv_{\min} - yw_{\min} \\ 0 & 0 & 1 \end{bmatrix}. \tag{6.7}$$

Матричное представление суммарного преобразования в нормированное поле просмотра имеет такой вид

$$\mathbf{M}_{\text{окно, норм. п. пр.}} = \mathbf{T} \cdot \mathbf{S} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{6.8}$$

что дает такой же результат, что и уравнения (6.3). Любую другую опорную точку отсекающего окна, например, правый верхний угол или центр окна, можно использовать в операциях масштабирования (трансляции). Кроме того, можно вначале транслировать точку в отсекающем окне в соответствующую точку поля просмотра, а затем масштабировать ее положение относительно положения поля просмотра.

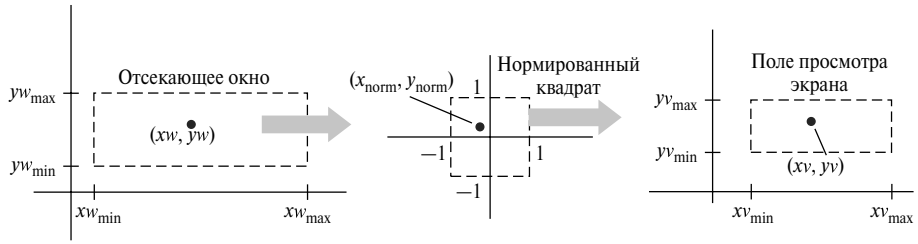


Рис. 6.8. Точка (xw, yw) в отсекающем окне переходит в точку с нормированными координатами $(x_{\text{norm}}, y_{\text{norm}})$, а затем — в точку с экранными координатами (xv, yv) в поле просмотра. Перед преобразованием в координаты поля просмотра объекты отсекаются единичным квадратом

Преобразование окна в поле просмотра сохраняет относительное расположение описаний объектов. Объект внутри отсекающего окна переводится в соответствующую точку внутри поля просмотра. Аналогично объект, внешний по отношению к отсекающему окну, будет внешним относительно поля просмотра.

С другой стороны, относительные пропорции объектов сохраняются, только если равны характеристические отношения поля просмотра и отсекающего окна. Другими словами, пропорции сохраняются, если масштабные коэффициенты sx и sy одинаковы. В противном случае внешние объекты будут растягиваться или сжиматься в направлении x или y (или обоих) при отображении на устройстве вывода.

Процедуры отсечения можно применить, используя либо границы отсекающего окна, либо границы поля просмотра. После отсечения нормированные координаты переводятся в координаты устройства. Кроме того, единичный квадрат может отображаться на устройство вывода с использованием таких же процедур, что и при преобразовании окна в поле просмотра, причем область внутри единичного квадрата переводится в полную область экрана устройства вывода.

ОТОБРАЖЕНИЕ ОТСЕКАЮЩЕГО ОКНА В НОРМИРОВАННЫЙ КВАДРАТ

Существует и другой подход к двухмерному наблюдению: вначале отсекающее окно преобразуется в нормированный квадрат, проводится отсечение в нормированных координатах, а затем описание сцены переводится в поле просмотра, заданное в экранных координатах. Данное преобразование иллюстрируется на рис. 6.8, где нормированные координаты меняются в диапазоне от -1 до 1. Алгоритм отсечения в данной последовательности преобразований теперь стандартизован, поэтому объекты вне границ $x = \pm 1$ и $y = \pm 1$ обнаруживаются и удаляются из описания сцены. На конечном этапе преобразования наблюдения объекты в поле просмотра размещаются в окне на экране дисплея.

Содержимое отсекающего окна переносится в единичный квадрат с использованием таких же процедур, что и при преобразовании окна в поле просмотра. Чтобы получить матрицу нормировки, нужно в уравнение (6.8) подставить -1 вместо $xv_{\min}$

и $yv_{\min}$ и $+1$ вместо $xv_{\max}$ и $yv_{\max}$. Затем вычисляются значения t_x, t_y, s_x и s_y, и получаем

$$\mathbf{M}_{\text{окно, норм. квадрат}} = \begin{bmatrix} \dfrac{2}{xw_{\max} - xw_{\min}} & 0 & -\dfrac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} \\ 0 & \dfrac{2}{yw_{\max} - yw_{\min}} & -\dfrac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} \\ 0 & 0 & 1 \end{bmatrix}. \tag{6.9}$$

Таким образом, после применения алгоритмов отсечения нормированный квадрат со стороной 2 переходит в заданное поле просмотра. На этот раз матрица преобразования получается из уравнения (6.8) подстановкой -1 вместо $xw_{\min}$ и $yw_{\min}$ и $+1$ вместо $xw_{\max}$ и $yw_{\max}$:

$$\mathbf{M}_{\text{окно, норм. квадрат}} = \begin{bmatrix} \dfrac{xv_{\max} - xv_{\min}}{2} & 0 & \dfrac{xv_{\max} + xv_{\min}}{2} \\ 0 & \dfrac{yv_{\max} - yv_{\min}}{2} & \dfrac{yv_{\max} + yv_{\min}}{2} \\ 0 & 0 & 1 \end{bmatrix}. \tag{6.10}$$

Последний этап процесса наблюдения — разместить область поля просмотра в окне на экране дисплея. Обычно левый нижний угол поля просмотра располагается в точке, заданной относительно левого нижнего угла окна на экране. На рис. 6.9 демонстрируется размещение поля просмотра в окне на экране дисплея.

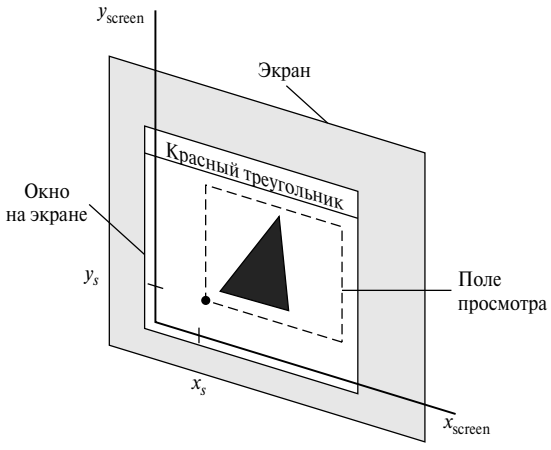


Рис. 6.9. Поле просмотра начинается в точке с координатами (x_s, y_s) окна на экране дисплея

Как и ранее, для сохранения исходных пропорций объектов характеристические отношения поля просмотра и отсекающего окна выбираются равными. В противном случае объекты будут растягиваться или сжиматься в направлении x или y. Кроме того, характеристическое отношение окна на экране дисплея может влиять на пропорции объектов. Если поле просмотра отображается на все окно на экране дисплея, и размер окна на экране дисплея меняется, объекты могут искажаться, если соответствующим образом не изменить характеристическое отношение поля просмотра.

ИЗОБРАЖЕНИЕ СТРОК СИМВОЛОВ

Существует два способа обработки строк символов, когда они "проходят по конвейеру наблюдения" к полю просмотра. Простейший метод сохраняет постоянный размер символов. Этот метод можно реализовать с растровыми символами. В то же время, наравне с другими примитивами можно преобразовывать и эскизные шрифты; достаточно преобразовать определенные положения отрезков в формах эскизных символов. Затем при обработке других примитивов сцены применяются алгоритмы определения точечных изображений преобразованных символов.

РАСЩЕПЛЕНИЕ ЭКРАНА И НЕСКОЛЬКО УСТРОЙСТВ ВЫВОДА

Выбирая на сцене различные отсекающие окна и соответствующие поля просмотра, можно одновременно отображать несколько объектов, несколько частей изображения или различные проекции одной сцены. Кроме того, эти проекции можно располагать в различных частях одного окна на экране дисплея или в нескольких окнах на экране. В конструкторских приложениях, например, можно отображать каркасную проекцию объекта в одном поле просмотра, при этом выводя на экран в другом поле просмотра полностью визуализированное изображение объекта. Кроме того, в третьем поле просмотра можно указывать другую информацию или выводить меню.

Помимо этого, можно параллельно использовать несколько устройств вывода в одной системе, и для каждого устройства можно задать пару "отсекающее окно-поле просмотра". Отображение на выбранное устройство вывода иногда называется *преобразованием рабочей станции*. В этом случае поля просмотра могут задаваться в координатах конкретного устройства вывода, или же каждое поле просмотра можно задавать в единичном квадрате, который затем отображается на выбранное устройство вывода. В некоторых графических системах для этого применяется пара функций рабочей станции. Одна функция используется для задания отсекающего окна для выбранного устройства вывода, определяемого *числом рабочей станции*, а другая — для задания поля просмотра, соответствующего этому устройству.

6.4. ФУНКЦИИ ДВУХМЕРНОГО НАБЛЮДЕНИЯ OpenGL

В действительности стандартная библиотека OpenGL не имеет функций, специально предназначенных для двухмерного наблюдения, поскольку данный пакет разработан преимущественно для трехмерных приложений. Однако основная библиотека OpenGL содержит функцию поля просмотра, а существующие процедуры трехмерного наблюдения можно использовать на двухмерной сцене. Кроме того, OpenGL Utility (GLU) содержит двухмерную функцию для задания отсекающего окна, а GLUT — функции, предназначенные для обработки окон на экране дисплея. Следовательно, данные двухмерные процедуры и функцию OpenGL поля просмотра можно использовать во всех необходимых операциях наблюдения.

РЕЖИМ ПРОЕКТИРОВАНИЯ OpenGL

Прежде чем выбирать отсекающее окно и поле просмотра в OpenGL, нужно установить подходящий режим построения матрицы преобразования из внешних координат в экранные. С помощью OpenGL нельзя задать отдельную двухмерную систему наблюдения, как показано на рис. 6.4, и как часть проективного преобразования нужно указать параметры отсекающего окна. Следовательно, вначале нужно выбрать режим проектирования. Для этого используется та же функция, которая применялась для установки режима проекции модели для геометрических преобразований. Затем к проекционной матрице применяются следующие команды, определяющие отсекающее окно и поле просмотра.

```
glMatrixMode (GL_PROJECTION);
```

Указанная команда присваивает проекционной матрице статус текущей (изначально текущей является единичная матрица). Впрочем, если предполагается отменить данную команду, например, чтобы получить другую проекцию сцены, инициализацию можно также проводить с помощью команды

```
glLoadIdentity ( );
```

Это гарантирует, что при каждом переходе в режим проектирования матрица становится единичной, так что новые параметры наблюдения не будут объединяться с предыдущими.

ФУНКЦИЯ ОТСЕКАЮЩЕГО ОКНА GLU

Чтобы определить двухмерное отсекающее окно, можно использовать функцию OpenGL Utility:

```
gluOrtho2D (xwmin, xwmax, ywmin, ywmax);
```

Координаты границ отсекающего окна указываются как числа с двойной точностью. Данная функция задает ортогональную проекцию для отображения сцены на экран. Для трехмерной сцены это означает, что объекты будут проектироваться вдоль параллельных линий, которые перпендикулярны двухмерному экрану xy. Однако в двухмерных приложениях объекты уже определены на плоскости xy. Следовательно, ортогональная проекция не влияет на нашу двухмерную сцену иным образом, кроме преобразования точек объекта в нормированные координаты. Тем не менее ортогональную проекцию нужно задать, поскольку наша двухмерная сцена обрабатывается по полному трехмерному конвейеру наблюдения OpenGL. Фактически отсекающее окно можно задать с использованием трехмерной версии функции `gluOrtho2D` из корневой библиотеки OpenGL (раздел 7.10).

В процедурах отсечения OpenGL используются нормированные координаты в диапазоне от -1 до 1. Функция `gluOrtho2D` задает трехмерную версию матрицы преобразования (6.9) для отображения объектов в отсекающем окне в нормированные координаты. Объекты вне нормированного квадрата (и вне отсекающего окна) удаляются с отображаемой сцены.

Если в программе-приложении не задать отсекающее окно, будут использоваться координаты по умолчанию $(xw_{\min}, yw_{\min}) = (-1,0; -1,0)$ и $(xw_{\max}, yw_{\max}) = (1,0; 1,0)$. Следовательно, по умолчанию отсекающее окно — это нормированный квадрат с центром в начале координат и стороной 2.

ФУНКЦИЯ ПОЛЯ ПРОСМОТРА OpenGL

Параметры поля просмотра задаются с помощью следующей функции OpenGL.

```
glViewport (xvmin, yvmin, vpWidth, vpHeight);
```

Все значения параметров даются в целочисленных экранных координатах относительно окна на экране дисплея. Параметры `xvmin` и `yvmin` задают положение левого нижнего угла поля просмотра относительного левого нижнего угла окна на экране дисплея. Ширина и высота пикселя поля просмотра задаются параметрами `vpWidth` и `vpHeight`. Если в программе не вызывать функцию `glViewport`, по умолчанию размер и положение поля просмотра будут такими же, как у окна на экране дисплея.

После применения процедур отсечения, точки в нормированном квадрате переводятся в прямоугольник поля просмотра с использованием матрицы (6.10). Требуемые в данной матрице координаты правого верхнего угла поля просмотра вычисляются через ширину и высоту поля просмотра:

$$xv_{\max} = xv_{\min} + vpWidth, \qquad yv_{\max} = yv_{\min} + vpHeight. \tag{6.11}$$

Для окончательного преобразования в буфер регенерации в заданные положения загружаются цвета пикселей примитивов поля просмотра.

В OpenGL можно создавать несколько полей просмотра (раздел 6.3). Параметры текущего активного поля просмотра можно узнать, используя такую функцию запроса.

```
glGetIntegerv (GL_VIEWPORT, vpArray);
```

Здесь `vpArray` — четырехэлементный массив с одним индексом. Данная функция `Get` возвращает параметры текущего поля просмотра в `vpArray` в порядке `xvmin`, `yvmin`, `vpWidth` и `vpHeight`. В интерактивных приложениях, например, данную функцию можно использовать для получения параметров поля просмотра, которое содержит курсор экрана.

СОЗДАНИЕ ОКНА НА ЭКРАНЕ ДИСПЛЕЯ GLUT

В разделе 2.9 кратко представлялись некоторые функции библиотеки GLUT. Поскольку эта библиотека сопрягается с любой системой управления окнами, процедуры GLUT используются для создания окон на экране дисплея и работы с ними, кроме того, приводимые примеры программ будут независимы от конкретной машины. Чтобы получить доступ к этим процедурам, вначале нужно инициализировать GLUT с помощью следующей функции.

```
glutInit (&argc, argv);
```

Параметры этой функции инициализации те же, что и у процедуры `main`, и `glutInit` можно использовать для обработки аргументов командной строки.

В GLUT есть три функции для определения окна на экране дисплея и выбора его размерности и положения:

```
glutInitWindowPosition (xTopLeft, yTopLeft);
glutInitWindowSize (dwWidth, dwHeight);
glutCreateWindow ("Title of Display Window");
```

Первая из этих функций дает целое число — положение в экранных координатах левого верхнего угла окна на экране дисплея относительно левого нижнего угла экрана. Если какая-то из координат отрицательна, точка окна на экране дисплея определяется системой управления окнами. С помощью второй функции выбирается ширина и высота окна на экране дисплея в пикселях (положительное целое число). Если данные функции задания размера и положения не используются, по умолчанию размер равен 300 на 300, а положение — $(-1, -1)$, и окно на экране дисплея размещает система управления окнами. В любом случае размер и положение окна на экране, заданное с помощью процедур GLUT, можно проигнорировать в зависимости от состояния или других требований, активных в данное время в системе управления окнами. Следовательно, система окон может выбирать место и устанавливать размер окна по-разному. Третья из приведенных функций создает окно на экране дисплея с

заданным размером и положением и присваивает название, хотя использование заголовка также зависит от системы окон. В момент обработки данной команды окно определяется, но не выводится на экран, пока не будут завершены все операции настройки GLUT.

УСТАНОВКА РЕЖИМА И ЦВЕТА ОКНА (GLUT)

С помощью приведенной ниже функции GLUT выбираются различные параметры окна на экране дисплея.

```
glutInitDisplayMode (mode);
```

Данная функция используется для выбора цветового режима (RGB или индексация) и различных комбинаций буферов. Выбранные параметры объединяются логической операцией ИЛИ. Режим по умолчанию — простая буферизация (один буфер) и цветовой режим RGB (или RGBA), и этот режим можно задать следующим объявлением:

```
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
```

Спецификация цветового режима `GLUT_RGB` эквивалентна `GLUT_RGBA`. Цвет фона для окна на экране дисплея выбирается в режиме RGB с помощью процедуры OpenGL

```
glClearColor (red, green, blue, alpha);
```

В режиме цветового индекса цвет окна на экране дисплея задается командой

```
glClearIndex (index);
```

Здесь параметру `index` присваивается значение, соответствующее положению в таблице цветов.

ИДЕНТИФИКАТОР GLUT ОКНА ИЗОБРАЖЕНИЯ

При необходимости в приложении можно создать несколько окон на экране дисплея, каждому из которых присваивается положительный целый *идентификатор окна изображения*, начиная со значения 1 для первого созданного окна. В момент инициации окна изображения его идентификатор можно записать с помощью выражения

```
windowID = glutCreateWindow ("A Display Window");
```

После того как целый идентификатор окна изображения записан в имени переменной `windowID`, это число можно использовать для изменения параметров дисплея или удаления окна с дисплея.

УДАЛЕНИЕ ОКНА GLUT С ЭКРАНА ДИСПЛЕЯ

Библиотека GLUT также включает функцию для удаления созданного окна с экрана. Если известен идентификатор окна, оно может удаляться с помощью такой команды.

```
glutDestroyWindow (windowID);
```

ТЕКУЩЕЕ ОКНО НА ЭКРАНЕ GLUT

Если задана операция с окном на экране, она применяется к *текущему окну*, которым является либо последнее созданное окно на экране, либо окно, выбранное с помощью следующей команды.

```
glutSetWindow (windowID);
```

Кроме того, в любой момент можно запросить систему и определить, какое окно является текущим:

```
currentWindowID = glutGetWindow ( );
```

Значение 0 возвращается этой функцией, если не существует окон на экране или если текущее окно на экране было удалено.

ЗАМЕЩЕНИЕ И ИЗМЕНЕНИЕ РАЗМЕРА ОКНА НА ЭКРАНЕ (GLUT)

Изменить положение текущего окна на экране можно с помощью команды

```
glutPositionWindow (xNewTopLeft, yNewTopLeft);
```

Здесь координаты задают новое положение левого верхнего угла окна изображения относительно левого верхнего угла экрана. Подобным образом следующая функция обновляет размер текущего окна на экране.

```
glutReshapeWindow (dwNewWidth, dwNewHeight);
```

Наконец, с помощью следующей команды можно расширить текущее окно на весь экран.

```
glutFullScreen ( );
```

Точный размер окна на экране после выполнения данной процедуры зависит от системы управления окнами. Кроме того, последующий вызов функции `glutPositionWindow` или `glutReshapeWindow` аннулирует запрос на расширение до полноэкранного размера.

При любом изменении размера окна может поменяться его характеристическое отношение, и объекты будут искажены по сравнению с первоначальными формами. Как отмечалось в разделе 3.24, изменение размеров окна на экране можно компенсировать с помощью следующего выражения.

```
glutReshapeFunc (winReshapeFcn);
```

Данная процедура GLUT активизируется при изменении размера окна на экране, и новая ширина и высота передаются аргументу этой процедуры — в данном примере это функция `winReshapeFcn`. Следовательно, `winReshapeFcn` можно назвать "функцией обратного вызова" для "события изменения формы". Затем данную функцию можно применить для обратного вызова, так изменив параметры поля просмотра, чтобы сохранялось исходное характеристическое отношение сцены. Кроме того, можно изменить границы отсекающего окна, цвет окна на экране, согласовать другие параметры наблюдения и выполнить другие задачи.

УПРАВЛЕНИЕ НЕСКОЛЬКИМИ ОКНАМИ НА ЭКРАНЕ (GLUT)

Библиотека GLUT также содержит несколько процедур для различных манипуляций с окнами на экране. Данные процедуры особенно полезны, когда на экране есть несколько окон, и их нужно переупорядочить или расположить в нужном месте экрана определенное окно.

Для преобразования текущего окна на экране в пиктограмму (небольшое изображение или символ, представляющий окно) используется следующая процедура.

```
glutIconifyWindow ( );
```

Данная пиктограмма помечается тем же именем, что присвоено окну, но имя пиктограммы можно изменить, используя такую команду:

```
glutSetIconTitle ("Icon Name");
```

Кроме того, можно изменить имя окна на экране, используя похожую команду:

```
glutSetWindowTitle ("New Window Name");
```

Когда на экране открыто несколько окон, часть из них может перекрываться или полностью закрывать другие окна экрана. Любое окно можно вывести перед всеми остальными, вначале сделав его текущим, а затем выполнив команду "всплывания".

```
glutSetWindow (windowID); glutPopWindow ( );
```

Подобным образом можно "запихнуть" текущее окно назад, чтобы оно располагалось позади всех окон экрана. В этом случае используется такая последовательность операций.

```
glutSetWindow (windowID); glutPushWindow ( );
```

Кроме того, текущее окно можно убрать с экрана, использовав команду

```
glutHideWindow ( );
```

Также можно вернуть “скрытое” окно экрана или окно, превращенное в пиктограмму, сделав его текущим, а затем вызвав следующую функцию.

```
glutShowWindow ( );
```

СУБОКНА GLUT

В выбранном окне экрана можно задать любое количество окон второго уровня, называемых *субокнами*. Это позволяет разбивать окна на различные участки. Субокно создается следующей функцией.

```
glutCreateSubWindow (windowID, xBottomLeft, yBottomLeft,
                        width, height);
```

Параметр `windowID` идентифицирует окно на экране, в котором требуется задать субокно. Остальные параметры задают размер субокна и расположение его левого нижнего угла относительно левого нижнего угла окна.

Субокнам присваивается положительный целый идентификатор так же, как нумеруются окна первого уровня. Кроме того, субокно можно расположить внутри другого субокна. К тому же, с каждым субокном можно соотнести отдельный режим отображения и другие параметры. Можно даже менять форму, положение субокон, помещать на передний или задний план, скрывать и показывать точно так же, как окна первого уровня. Однако субокно GLUT нельзя превратить в пиктограмму.

ВЫБОР ФОРМЫ КУРСОРА ОКНА НА ЭКРАНЕ

Чтобы придать форму экранному курсору, используемому в текущем окне, можно использовать следующую процедуру GLUT.

```
glutSetCursor (shape);
```

Возможны такие формы курсоров: стрелка, показывающая в выбранном направлении, двунаправленная стрелка, вращающаяся стрелка, перекрестье, наручные часы, вопросительный знак и даже череп и скрещенные кости. Например, можно присвоить параметру `shape` значение (символьная константа) `GLUT_CURSOR_UP_DOWN` и получить двунаправленную (вверх и вниз) стрелку. Вращающаяся стрелка выбирается с помощью константы `GLUT_CURSOR_CYCLE`, форма наручных часов выбирается с помощью `GLUT_CURSOR_WAIT`, а череп и скрещенные кости — это константа `GLUT_CURSOR_DESTROY`. Форму курсора можно соотнести с окном экрана, чтобы указать определенный тип приложения, например, анимацию. Однако точные формы, которые можно использовать, зависят от системы.

НАБЛЮДЕНИЕ ГРАФИЧЕСКИХ ОБЪЕКТОВ В ОКНЕ ЭКРАНА GLUT

После создания окна на экране и выбора его положения, размера, цвета и других характеристик указывается, что будет показано в этом окне. Если на экране было создано несколько окон, требуемое окно вначале обозначается как текущее. Затем, чтобы присвоить что-то этому окну, вызывается следующая функция.

```
glutDisplayFunc (pictureDescrip);
```

Аргумент — это процедура, описывающая, *что* будет изображено в текущем окне. Данная процедура (в приведенном примере — с именем `pictureDescrip`) называется *функцией обратного вызова*, поскольку она будет выполняться, когда GLUT решит, что содержимое окна на экране нужно обновить. Процедура `pictureDescrip` обычно содержит примитивы и параметры OpenGL, определяющие изображение, хотя она может задавать другие конструкции, такие как изображение меню.

Если на экране задано несколько окон, описанный процесс повторяется для каждого окна или субокна. Кроме того, может потребоваться вызвать `glutDisplayFunc` после команды `glutPopWindow`, если окно на экране было повреждено при повторном изображении окон. В этом случае используется следующая функция, чтобы указать, что содержимое текущего окна на экране нужно обновить.

```
glutPostRedisplay ( );
```

Данная процедура также используется, когда в окне на экране нужно показать такой дополнительный объект, как всплывающее меню.

ВЫПОЛНЕНИЕ ПРИКЛАДНОЙ ПРОГРАММЫ

Когда настройка программы завершена, и окна на экране созданы и инициализированы, нужно выполнить конечную команду GLUT, которая сообщает о выполнении программы.

```
glutMainLoop ( );
```

В этот момент окна экрана и их графическое содержимое посылаются на экран. Программа также вводит *цикл обработки GLUT*, который постоянно проверяет на наличие новых “событий”, таких как интерактивный ввод мышью или ввод с графического планшета.

ДРУГИЕ ФУНКЦИИ GLUT

Библиотека GLUT предлагает множество разнообразных процедур обработки процессов, зависящих от системы, дополняя основную библиотеку OpenGL. Например, данная библиотека содержит функции для генерации растровых и эскизных символов (раздел 3.21), и она предоставляет функции загрузки значений в таблицу цветов (раз-

дел 4.3). Кроме того, некоторые функции GLUT, рассмотренные в главе 8, позволяют отображать трехмерные объекты либо как объемные тела, либо в форме каркасного представления. Данные объекты включают сферу, тор, пять правильных многогранников (куб, тетраэдр, октаэдр, додекаэдр и икосаэдр).

Иногда бывает удобно задать такую функцию, которая будет выполняться при отсутствии иных событий, которые обрабатывает система. Это делается так.

```
glutIdleFunc (function);
```

Параметром этой процедуры GLUT может быть фоновая функция или процедура, обновляющая параметры анимации, когда нет других процессов.

Кроме того, существуют функции GLUT, рассмотренные в главе 11, которые предназначены для получения и обработки интерактивного ввода, а также создания меню и управления ими. GLUT предлагает отдельные процедуры для разных устройств ввода, таких как мышь, клавиатура, графический планшет, пространственный манипулятор (spaceball).

Наконец, можно использовать следующую функцию для запроса системы о некоторых текущих параметрах состояния.

```
glutGet (stateParam);
```

Данная функция возвращает целое значение, соответствующее символьной константе, выбранной в качестве аргумента. Например, можно получить координату x левого верхнего угла текущего окна на экране относительного левого верхнего угла экрана, использовав константу `GLUT_WINDOW_X`. Можно также получить текущую ширину окна или экрана, использовав константу `GLUT_WINDOW_WIDTH` или `GLUT_SCREEN_WIDTH`.

ПРИМЕР ПРОГРАММЫ ДВУХМЕРНОГО НАБЛЮДЕНИЯ OpenGL

Продемонстрируем использование функции поля просмотра OpenGL, расщепив экран, чтобы показать две проекции треугольника в плоскости xy, центр масс которого находится в начале внешней системы координат. Изначально поле просмотра определяется в левой половине окна на экране, здесь же синим цветом отображается исходный треугольник. Затем для того же самого отсекающего окна определяется другое поле просмотра для правой половины окна, а цвет заполнения меняется на красный. Затем треугольник вращается относительно центра масс и отображается во втором поле просмотра. Оба треугольника, выводимые на экран данной программой, показаны на рис. 6.10.

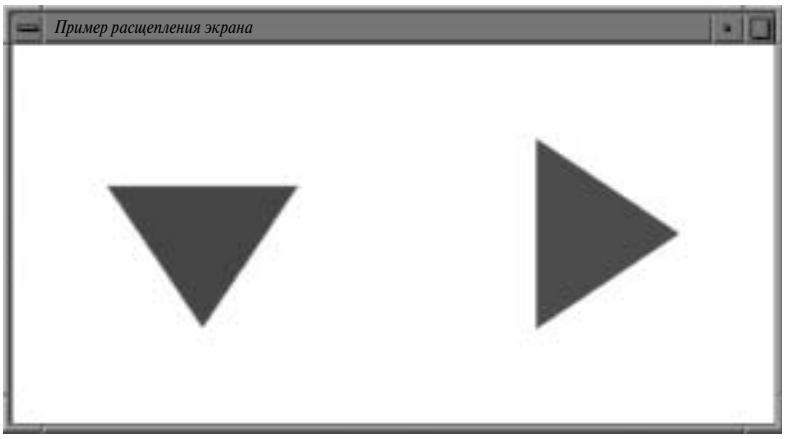


Рис. 6.10. Расщепление экрана в окне с помощью процедуры `displayFcn`

```
#include <GL/glut.h> class wcPt2D {
   public:
      GLfloat x, y; };
void init (void) {
   /* Задается белый цвет окна на экране. */
   glClearColor (1.0, 1.0, 1.0, 0.0);
   /* Задаются параметры отсекающего окна во внешних
      координатах.                                        */
   glMatrixMode (GL_PROJECTION);
   gluOrtho2D (-100.0, 100.0, -100.0, 100.0);
   /* Задается режим построения геометрической матрицы
      преобразования.                                      */
   glMatrixMode (GL_MODELVIEW); }
void triangle (wcPt2D *verts) {
   GLint k;
   glBegin (GL_TRIANGLES);
      for (k = 0; k < 3; k++)
         glVertex2f (verts [k].x, verts [k].y);
   glEnd ( ); }
void displayFcn (void) {
   /*  Определяется исходное положение треугольника. */
   wcPt2D verts [3] = { {-50.0, -25.0}, {50.0, -25.0},
                                {0.0, 50.0} };
   glClear (GL_COLOR_BUFFER_BIT);            // Очищается окно.
   glColor3f (0.0, 0.0, 1.0);       // Цвет заполнения - синий.
   glViewport (0, 0, 300, 300);
   /* Задается левое поле просмотра. */
   triangle (verts);               // Отображается треугольник.
   /* Треугольник поворачивается и отображается в правой
    * половине окна.                                        */
```

```
    glColor3f (1.0, 0.0, 0.0);     // Цвет заполнения - красный.
    glViewport (300, 0, 300, 300);
    /* Задается правое поле просмотра. */
    glRotatef (90.0, 0.0, 0.0, 1.0);    // Поворот вокруг оси z.
    triangle (verts);
    /* Отображается красный повернутый треугольник. */
    glFlush ( );
}
void main (int argc, char ** argv) {
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition (50, 50);
    glutInitWindowSize (600, 300);
    glutCreateWindow ("Пример расщепления экрана");
    init ( );
    glutDisplayFunc (displayFcn);
    glutMainLoop ( );
}
```

6.5. АЛГОРИТМЫ ОТСЕЧЕНИЯ

В общем случае любая процедура, которая удаляет те участки изображения, которые находятся внутри или снаружи заданной области пространства, называется *алгоритмом отсечения* или просто *отсечением*. Обычно отсекающая область — это прямоугольник стандартной ориентации, хотя в алгоритме отсечения можно использовать любую форму.

Чаще всего отсечение применяется в конвейере наблюдения, где служит для извлечения обозначенного участка сцены (двух- или трехмерной) с целью отображения на устройстве вывода. Методы отсечения также используются для защиты от наложения границ объектов, построения объектов с использованием методов объемного моделирования, управления средой с несколькими окнами и перемещения, копирования или стирания участков изображения в различных программах рисования.

Алгоритмы отсечения применяются к двухмерным процедурам наблюдения, чтобы определить те части изображения, которые находятся внутри отсекающего окна. Затем все, что находится вне отсекающего окна, удаляется из описания сцены, передаваемого на устройство вывода для отображения. Для эффективной реализации отсечения в конвейере наблюдения алгоритмы применяются к нормированным границам отсекающего окна. Это сокращает расчеты, поскольку все матрицы геометрических преобразований и преобразований наблюдения можно объединить и применить к описанию сцены перед отсечением. Обрезанную сцену можно затем перевести в экранные координаты для окончательной обработки.

В следующих разделах исследуются такие двухмерные алгоритмы.

- Отсечение точки.
- Отсечение линии (прямых отрезков).
- Отсечение закрашенной области (многоугольники).
- Отсечение кривой.
- Отсечение текста.

Отсечения точки, линии и многоугольника являются стандартными компонентами графических пакетов. Однако схожие методы можно применить и к другим объектам, в частности, коническим сечениям, эллипсам и сферам, а также сплайновым кривым и поверхностям. Обычно, однако, для сокращения вычислений объекты с нелинейными границами аппроксимируются прямыми отрезками или многоугольными поверхностями.

Если не оговорено иное, будем предполагать, что отсекающая область — это прямоугольное окно стандартной ориентации, углы которого находятся в точках с координатами $xw_{\min}$, $xw_{\max}$, $yw_{\min}$ и $yw_{\max}$. Данные углы обычно соответствуют нормированному квадрату, в котором значения x и y принадлежат диапазону от 0 до 1 или от -1 до 1.

6.6. ДВУХМЕРНОЕ ОТСЕЧЕНИЕ ТОЧКИ

Для двухмерного прямоугольника стандартной ориентации двухмерная точка $\mathbf{P} = (x, y)$ извлекается для отображения, если удовлетворяется следующее неравенство:

$$\begin{aligned} xw_{\min} \le x \le xw_{\max}, \\ yw_{\min} \le y \le yw_{\max}. \end{aligned} \tag{6.12}$$

Если любое из этих неравенств не удовлетворяется, точка отсекается (не сохраняется для отображения).

Хотя отсечение точки применяется реже, чем отсечение линии или многоугольника, в различных ситуациях оно полезно, особенно когда изображения смоделированы системой многих частиц. Например, отсечение точки можно применить к сценам, содержащим облака, морскую пену, дым или взрывы, которые моделируются “частицами” (координатами центров маленьких окружностей или сфер).

6.7. ДВУХМЕРНОЕ ОТСЕЧЕНИЕ ЛИНИИ

На рис. 6.11 иллюстрируются возможные положения прямых отрезков относительно стандартного отсекающего окна. Алгоритм отсечения линии обрабатывает каждую

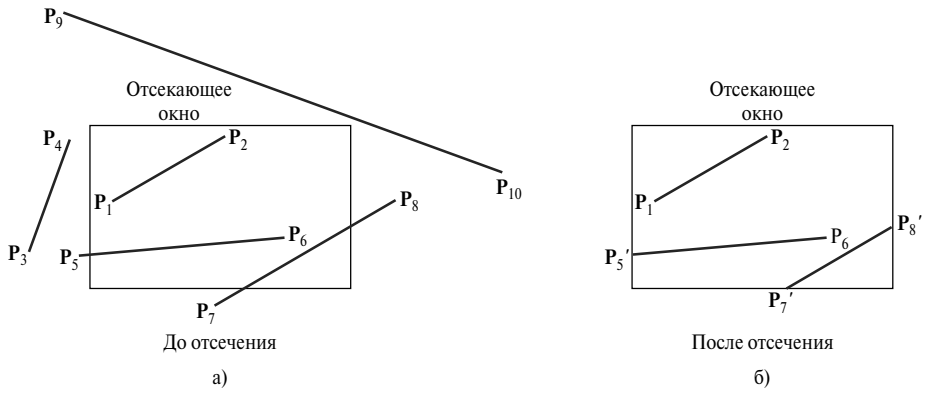


Рис. 6.11. Отсечение прямых отрезков с использованием стандартного прямоугольного отсекающего окна

линию на сцене с помощью последовательности проверок и расчетов точек пересечения, позволяющих определить, нужно ли записывать линию или ее часть. Ресурсоемкой частью процедуры отсечения линии является расчет точек пересечения линии со сторонами окна. Следовательно, главная цель любого алгоритма отсечения линии — минимизировать расчет точек пересечения. Чтобы это сделать, можно вначале выполнить проверку и определить, не находится ли участок линии полностью внутри или полностью снаружи отсекающего окна. Определить, что линия полностью внутри отсекающего окна, легко, гораздо сложнее идентифицировать все линии, полностью лежащие за окном. Если мы не можем отнести линию к полностью внутренним или полностью внешним, нужно вычислить точки пересечения линии с периметром окна.

Чтобы проверить, находится ли прямой отрезок полностью внутри или полностью снаружи выбранной стороны отсекающего окна, проверяется возможность отсечения точки (см. предыдущий раздел). Если все концы отрезка находятся внутри всех четырех сторон отсекающего прямоугольника, как для линии от $\mathbf{P}_1$ до $\mathbf{P}_2$ на рис. 6.11, линия полностью лежит внутри отсекающего окна, и она сохраняется. Если оба конца отрезка находятся вне всех четырех сторон (линия $\overline{\mathbf{P}_3\mathbf{P}_4}$ на рис. 6.11), данный отрезок находится полностью вне окна, и он удаляется из описания сцены. Однако, если ни одна проверка не дала положительного результата, отрезок пересекает по меньшей мере одну сторону отсекающего прямоугольника и может пересекать или не пересекать внутреннюю часть отсекающего окна.

Чтобы сформулировать уравнение для прямого отрезка, используем следующее параметрическое представление, где точки с координатами $(x_0,\ y_0)$ и $(x_{\text{end}},\ y_{\text{end}})$ обозначают два конца отрезка.

$$\begin{aligned} x &= x_0 + u(x_{\text{end}} - x_0), \\ y &= y_0 + u(y_{\text{end}} - y_0), \qquad 0 \le u \le 1. \end{aligned} \tag{6.13}$$

На основе приведенного параметрического представления можно определить, где отрезок проходит через стороны отсекающего окна, подставив вместо x или y координаты и найти из уравнения параметр u. Например, левая граница окна проходит по $xw_{\min}$, поэтому подставим это значение вместо x, найдем u и вычислим соответствующую координату y точки пересечения. Если это значение u не входит в диапазон от 0 до 1, отрезок не пересекает данную границу окна. Однако, если значение u принадлежит диапазону 0–1, часть линии находится внутри границ. Затем можно обработать данную внутреннюю часть отрезка вместе с другими границами окна, пока не будет удалена вся линия, или пока не будет найден отрезок, полностью лежащий внутри окна.

Обработка отрезков на сцене с использованием простого подхода отсечения, описанного в предыдущем абзаце, является прямолинейной, но не очень эффективной. Вообще, исходную проверку и расчет точек пересечения можно переформулировать так, чтобы сократить время обработки набора прямых отрезков, и в настоящее время разработано несколько более быстрых схем отсечения линий. Некоторые из этих алгоритмов явно разработаны для двухмерных изображений, некоторые легко адаптируются для наборов трехмерных отрезков.

ОТСЕЧЕНИЕ ЛИНИЙ КОЭНА–САЗЕРЛЕНДА

Одним из первых алгоритмов, разработанных для быстрого отсечения линий, является схема Коэна–Сазерленда (Cohen–Sutherland), и разновидности этого метода очень широко используются. Время обработки в этом методе сокращено за счет большего числа проверок перед обработкой (нахождением точек пересечения). Изначально всем конечным точкам линий на изображении присваивается четырехзначное двоичное значение, называемое *кодом области*, и каждый двоичный разряд указывает, находится точка внутри или вне одной границы отсекающего окна. Границы окна можно пронумеровать в любом порядке, и на рис. 6.12 иллюстрируется одно возможное упорядочение, когда двоичные разряды нумеруются от 1 до 4 слева направо. Следовательно, при таком упорядочении крайний справа разряд (бит 1) соответствует левой границе отсекающего окна, а крайний слева разряд (бит 4) соответствует верхней границе окна. Значение 1 (или *true*) в любом разряде указывает, что конечная точка находится вне данной границы окна. Аналогично значение 0 (или *false*) в любом разряде указывает, что конечная точка не находится снаружи (внутри или на границе) соответствующей стороны окна. Иногда код области называется "*командой на выключение*", поскольку значение 1 в любом разряде указывает, что пространственная точка находится вне соответствующей границы отсечения.

Каждая сторона отсекающего окна делит двухмерное пространство на внутреннюю половину пространства и внешнюю. Всего четыре границы окна формируют девять областей, а на рис. 6.13 перечислены значения двоичного кода во всех этих областях. Следовательно, конечной точке, которая находится ниже и слева от отсека-

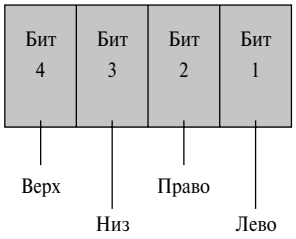


Рис. 6.12. Возможное упорядочение границ отсекающих окон, соответствующих двоичным разрядам кода области Коэна–Сазерленда

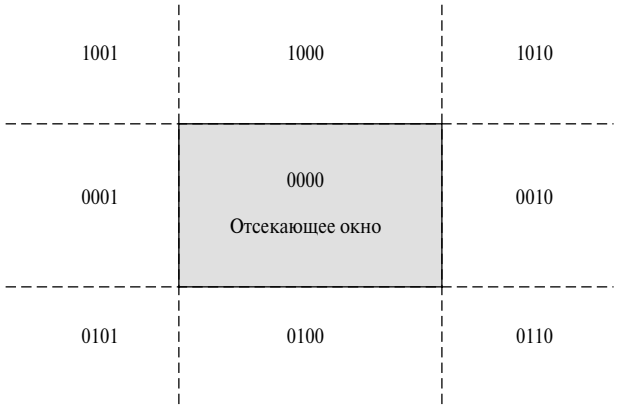


Рис. 6.13. Девять двоичных кодов области, идентифицирующих положение конечной точки относительно границ отсекающего окна

ющего окна, присваивается код области 0101, а код области любой конечной точки, находящейся внутри отсекающего окна, — 0000.

Двоичные значения кода области определяются сравнением координат (x, y) конечной точки с границами отсекающего прямоугольника. Бит 1 устанавливается, если $x < xw_{\min}$, подобным образом определяются значения трех остальных битов. Вместо проверки с помощью неравенства, эффективнее применять операции обработки битов и следующие два этапа: 1) рассчитать разности между координатами конечных точек и границами окна отсечения; 2) использовать знак результата каждой разности для задания соответствующего значения кода области. Для упорядочения, показанного на рис. 6.12, бит 1 является битом знака выражения $x - xw_{\min}$; бит 2 — бит знака выражения $xw_{\max} - x$; бит 3 — бит знака выражения $y - yw_{\min}$; и бит 4 — это бит знака выражения $yw_{\max} - y$.

После того как коды областей для всех конечных точек будут определены, можно быстро выявить, какие линии полностью лежат внутри окна, а какие очевидно лежат снаружи. Конечные точки всех линий, которые целиком вмещаются в окно, имеют код области 0000, и эти отрезки записываются. Любая линия, конечные точки которой имеют 1 в одинаковых разрядах кода области, лежит полностью за пределами окна, и этот отрезок удаляется. Например, линия, одна конечная точка которой имеет код области 1001, а другая — 0101, целиком находится слева от отсекающего окна, что видно по значению 1 в первом разряде обоих кодов области.

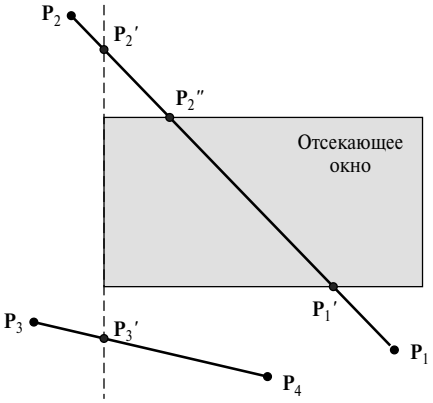


Рис. 6.14. Линии, проходящие из одной области отсекающего окна в другую, могут пересекать окно или пересекать одну или несколько границ отсечения, не входя в окно

Проверки "внутри-снаружи" можно выполнять с использованием логических операторов. Если значение операции *ИЛИ*, примененной к двум кодам конечных точек, — *false* (0000), отрезок находится внутри отсекающего окна. Следовательно, данная линия записывается для отображения, и проверяется следующая линия в описании сцены. Если результат операции *И*, примененной к двум кодам конечных точек, — *true* (не 0000), линия полностью лежит вне отсекающего окна, и ее можно исключить из описания сцены.

Линии, которые с помощью проверок кодов области нельзя однозначно отнести к полностью внешним или полностью внутренним, далее проверяются на предмет пересечения с границами окон. Как показано на рис. 6.14, отрезки могут пересекать границы отсечения, не попадая внутрь окна. Следовательно, для отсечения отрезка может потребоваться вычислить несколько точек пересечения в зависимости от порядка, в котором обрабатываются границы отсечения. При обработке каждой стороны отсекающего окна отсекается кусок линии, а оставшаяся часть линии проверяется на пересечение с другими границами окон. Удаление участков продолжается, пока линия не будет полностью обрезана, или пока оставшаяся часть линии не будет полностью располагаться внутри отсекающего окна. Далее будем предполагать, что стороны окна обрабатываются в таком порядке: левая, правая, нижняя, верхняя. Чтобы определить, пересекает ли линия выбранную границу отсечения, можно проверить соответствующие разряды кодов областей конечных точек. Если одно из этих значений равно 1, а второе — 0, отрезок пересекает данную границу.

На рис. 6.14 показаны два отрезка, которые невозможно сразу отнести к полностью внутренним или полностью внешним по отношению к отсекающему окну. Коды областей отрезка, соединяющего $\mathbf{P}_1$ с $\mathbf{P}_2$, равны 0100 и 1001. Следовательно, $\mathbf{P}_1$ находится внутри левой границы отсечения, а $\mathbf{P}_2$ — вне этой границы. Затем вычисляется точка пересечения $\mathbf{P}'_2$, и вырезается участок линии от $\mathbf{P}_2$ до $\mathbf{P}'_2$. Оставшаяся часть линии находится внутри правой граничной линии, поэтому переходим к

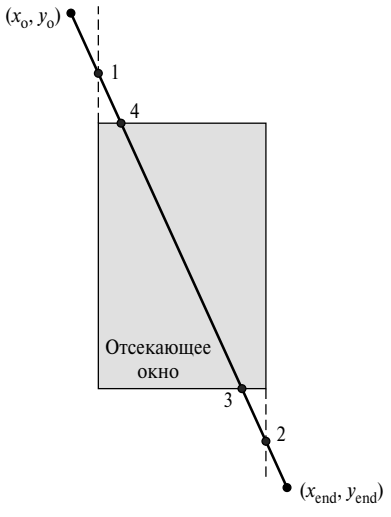


Рис. 6.15. Четыре точки пересечения (помечены цифрами от 1 до 4) отрезка, отсекаемого границами окна в таком порядке: левая, правая, нижняя, верхняя

проверке нижней границы. Конечная точка $\mathbf{P}_1$ расположена ниже отсекающей стороны, а $\mathbf{P}'_2$ — выше, поэтому далее находится точка пересечения с этой границей ($\mathbf{P}'_1$). Затем удаляется участок линии между $\mathbf{P}_1$ и $\mathbf{P}'_1$, и мы переходим к обработке верхнего края окна. Здесь определяется точка пересечения $\mathbf{P}''_2$. Заключительный этап — вырезание участка над верхней границей и запись внутреннего сегмента от $\mathbf{P}'_1$ до $\mathbf{P}''_2$. Для второй линии находим, что точка $\mathbf{P}_3$ находится вне левой границы, а $\mathbf{P}_4$ — внутри. Следовательно, нужно найти точку пересечения $\mathbf{P}'_3$ и удалить отрезок между $\mathbf{P}_3$ и $\mathbf{P}'_3$. Проверяя коды областей конечных точек $\mathbf{P}'_3$ и $\mathbf{P}_4$, находим, что оставшаяся часть линии находится ниже отсекающего окна, и ее также можно удалить.

При отсечении отрезка с использованием данного подхода, возможно, придется вычислить точки пересечения со всеми четырьмя границами отсечения в зависимости от того, как обрабатываются конечные точки линии и как упорядочены границы. На рис. 6.15 показаны четыре точки пересечения, которые можно вычислить для отрезка, который обрабатывается на предмет пересечения со сторонами отсекающего окна, идущими в следующем порядке: левая, правая, нижняя, верхняя. Поэтому были разработаны модификации данного алгоритма, направленные на сокращение расчетов точек пересечения.

Чтобы определить точки пересечения с отрезком, можно использовать уравнение прямой с угловым коэффициентом. Для прямой с координатами конечных точек $(x_0,\ y_0)$ и $(x_{\text{end}}, y_{\text{end}})$ координата y точки пересечения с вертикальной отсекающей границей находится следующим образом:

$$y = y_0 + m(x - x_0), \tag{6.14}$$

где значение x выбирается равным $xw_{\min}$ или $xw_{\max}$, а тангенс угла наклона прямой вычисляется как $m = (y_{\text{end}} - y_0)/(x_{\text{end}} - x_0)$. Подобным образом, если ищется точка пересечения с горизонтальной границей, координату x можно вычислить следующим образом:

$$x = x_0 + \frac{y - y_0}{m}, \tag{6.15}$$

где y равно $yw_{\min}$ или $yw_{\max}$.

Реализация двухмерного алгоритма отсечения прямых Коэна–Сазерленда представлена в следующих процедурах. Расширение этого алгоритма на трехмерную ситуацию очевидно, а более подробно трехмерное наблюдение рассматривается в следующей главе.

```
class wcPt2D {
   public:
      GLfloat x, y;
};
inline GLint round (const GLfloat a)
{ return GLint (a + 0.5); }

/* Для каждой внешней области прямоугольного отсекающего окна
 * определяется четырехбитовый код.                              */
 const GLint winLeftBitCode = 0x1;
 const GLint winRightBitCode = 0x2;
 const GLint winBottomBitCode = 0x4;
 const GLint winTopBitCode = 0x8;

/* Битовый код области также присваивается каждой конечной
 * точке входного отрезка согласно его расположению
 * относительно четырех краев входного прямоугольного
 * отсекающего окна.
 * Конечная точка с кодом области 0000 находится внутри
 * отсекающего окна, в противном случае она находится вне хотя
 * бы одной отсекающей границы. Если применение операции ИЛИ
 * к двум кодам конечных точек дает значение 'false', вся
 * линия, определенная этими двумя конечными точками
 * записывается (принимается). Если применение операции И к
 * двум кодам конечных точек дает значение 'true', линия лежит
 * полностью вне отсекающего окна, и она исключается из
 * дальнейшей обработки (отклоняется).                           */
inline GLint inside (GLint code) { return GLint (!code); }
inline GLint reject (GLint code1, GLint code2)
                        { return GLint (code1 & code2); }
inline GLint accept (GLint code1, GLint code2)
                        { return GLint (!(code1 | code2)); }
```

```
GLubyte encode (wcPt2D pt, wcPt2D winMin, wcPt2D winMax)
{
  GLubyte code = 0x00;
  if (pt.x < winMin.x)
    code = code | winLeftBitCode;
  if (pt.x > winMax.x)
    code = code | winRightBitCode;
  if (pt.y < winMin.y)
    code = code | winBottomBitCode;
  if (pt.y > winMax.y)
    code = code | winTopBitCode;
  return (code);
}
void swapPts (wcPt2D * p1, wcPt2D * p2)
{
wcPt2D tmp;
  tmp = *p1; *p1 = *p2; *p2 = tmp;
}
void swapCodes (GLubyte * c1, GLubyte * c2)
{
  GLubyte tmp;
  tmp = *c1; *c1 = *c2; *c2 = tmp;
}
void lineClipCohSuth (wcPt2D winMin, wcPt2D winMax,
                                wcPt2D p1, wcPt2D p2)
 {
  GLubyte code1, code2;
  GLint done = false, plotLine = false;
  GLfloat m;
  while (!done) {
    code1 = encode (p1, winMin, winMax);
    code2 = encode (p2, winMin, winMax);
    if (accept (code1, code2)) {
      done = true;
      plotLine = true;
    }
    else
      if (reject (code1, code2))
        done = true;
      else {
      /* Пометить конечную точку вне окна на экране как p1. */
        if (inside (code1)) {
          swapPts (&p1, &p2);
          swapCodes (&code1, &code2);
        }
        /*  Использовать тангенс угла наклона m */
        /*  для расчета пересечения линии со стороной. */
        if (p2.x != p1.x)
          m = (p2.y - p1.y) / (p2.x - p1.x);
        if (code1 & winLeftBitCode) {
```

```
            p1.y += (winMin.x - p1.x) * m;
            p1.x = winMin.x;
          }
          else
            if (code1 & winRightBitCode) {
              p1.y += (winMax.x - p1.x) * m;
              p1.x = winMax.x;
            }
            else
              if (code1 & winBottomBitCode) {
/* Нужно обновить p1.x только для невертикальных линий. */
                if (p2.x != p1.x)
                  p1.x += (winMin.y - p1.y) / m;
                p1.y = winMin.y;
              }
              else
                if (code1 & winTopBitCode) {
                  if (p2.x != p1.x)
                    p1.x += (winMax.y - p1.y) / m;
                  p1.y = winMax.y;
                }
        }
    }
    if (plotLine)
      lineBres (round (p1.x), round (p1.y), round (p2.x),
                                                round (p2.y));
}
```

ОТСЕЧЕНИЯ ЛИНИИ ЛИАНГА–БАРСКИ

Были разработаны и более быстрые алгоритмы отсечения линий, в которых перед расчетом точек пересечения производится больше проверок. Одна из первых работ в этом направлении принадлежит Сайресу (Cyrus) и Бэку (Beck), и она основана на анализе параметрических уравнений прямой. Позднее Лианг (Liang) и Барский (Barsky) независимо разработали даже еще более быструю форму параметрического алгоритма отсечения линий.

Для отрезка с конечными точками (x_0, y_0) и $(x_{\text{end}}, y_{\text{end}})$ линию можно описать в параметрической форме:

$$\begin{aligned} x &= x_0 + u\Delta x, \\ y &= y_0 + u\Delta y, \qquad 0 \le u \le 1, \end{aligned} \tag{6.16}$$

где $\Delta x = x_{\text{end}} - x_0$ и $\Delta y = y_{\text{end}} - y_0$. В алгоритме Лианга–Барски параметрические уравнения прямой объединяются с условиями отсечения линии (6.12), в результате

чего получаются неравенства

$$\begin{aligned} xw_{\min} &\le x_0 + u\Delta x \le xw_{\max}, \\ yw_{\min} &\le y_0 + u\Delta y \le yw_{\max}, \end{aligned} \tag{6.17}$$

которые можно выразить следующим образом:

$$u\ p_k \le q_k, \qquad k = 1, 2, 3, 4, \tag{6.18}$$

где параметры p и q определяются как

$$\begin{aligned} p_1 &= -\Delta x, & q_1 &= x_0 - xw_{\min}, \\ p_2 &= \Delta x, & q_2 &= xw_{\max} - x_0, \\ p_3 &= -\Delta y, & q_3 &= y_0 - yw_{\min}, \\ p_4 &= \Delta y, & q_4 &= yw_{\max} - y_0. \end{aligned} \tag{6.19}$$

Любая линия, параллельная одной из сторон отсекающего окна, имеет $p_k = 0$ при k, связанном с данной границей (k = 1, 2, 3 и 4 соотнесено с левой, правой, нижней и верхней границей соответственно). Если для этого значения k также будет $q_k < 0$, тогда линия полностью лежит вне границы, и ее можно исключить из дальнейшего рассмотрения. Если $q_k \ge 0$, линия находится внутри параллельной границы отсечения.

Если $p_k < 0$, бесконечное расширение линии идет снаружи внутрь бесконечного расширения определенной стороны отсекающего окна. Если $p_k > 0$, линия идет изнутри наружу. При ненулевом значении p_k можно вычислить величину u, которая соответствует точке, где бесконечно расширенная линия пересекает расширение края окна k:

$$u = \frac{q_k}{p_k}. \tag{6.20}$$

Для каждой линии можно вычислить значения параметров u_1 и u_2, которые определяют ту часть линии, которая лежит внутри отсекающего прямоугольника. Значение u_1 находится исследованием сторон прямоугольника, для которых линия идет снаружи внутрь ($p < 0$). Для этих сторон вычисляется $r_k = q_k/p_k$. Значение u_1 выбирается равным максимуму набора из 0 и различных значений r. И наоборот, значение u_2 определяется исследованием границ, для которых линия идет изнутри наружу ($p > 0$). Для каждой из этих границ вычисляется значение r_k, а значение u_2 равно минимуму набора, состоящего из 1 и рассчитанных значений r. Если $u_1 > u_2$, линия полностью находится снаружи отсекающего окна, и ее можно отбросить. В противном случае по двум значениям параметра u вычисляются конечные точки извлекаемой линии.

Описанный алгоритм реализован в следующем фрагменте кода. Параметры точек пересечения линий специализируются со значениями $u_1 = 0$ и $u_2 = 1$. Для каждой границы отсечения вычисляются подходящие значения p и q, которые затем используются функцией `clipTest` для определения, следует ли отбросить линию, или нужно вычислить параметры пересечения. Если $p < 0$, параметр r применяется для обновления u_1; при $p > 0$ параметр r используется для обновления u_2. Если обновление u_1 или u_2 приводит к $u_1 > u_2$, линия отбрасывается. В противном случае обновляется соответствующий параметр u, только если новое значение дает более короткую линию. Если $p = 0$ и $q < 0$, линию можно удалить, поскольку она параллельна границе и находится вне ее. Если линия не отброшена после проверки всех четырех значений p и q, по значениям u_1 и u_2 определяются конечные точки обрезанной линии.

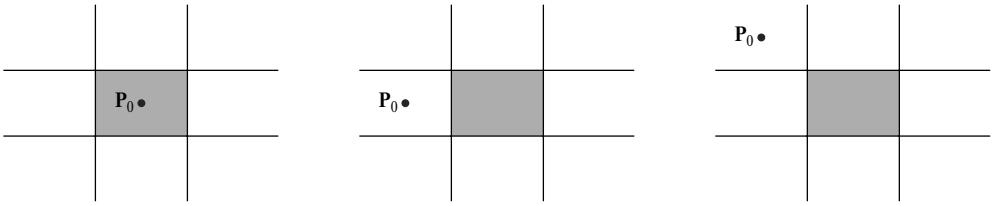
```
    class wcPt2D {
       private:
          GLfloat x, y;
       public:
       /* По умолчанию точка инициализируется как (0.0, 0.0). */
       wcPt3D ( ) {
          x = y = 0.0;
       }
       setCoords (GLfloat xCoord, GLfloat yCoord) {
          x = xCoord;
          y = yCoord;
       }
       GLfloat getx ( ) const {
          return x;
       }
       GLfloat gety ( ) const {
          return y;
       }
    };
inline GLint round (const GLfloat a)  {
    return GLint (a + 0.5);
}
GLint clipTest (GLfloat p, GLfloat q, GLfloat * u1,
                                          GLfloat * u2){
  GLfloat r;
  GLint returnValue = true;
  if (p < 0.0) {
    r = q / p;
    if (r > *u2)
      returnValue = false;
    else
      if (r > *u1)
        *u1 = r; }
```

```
      else
        if (p > 0.0) {
          r = q / p;

          if (r < *u1)
            returnValue = false;
          else if (r < *u2)
            *u2 = r;
        }
        else
          /* В этом случае p = 0, и линия параллельна границе
             отсечения.                                          */
          if (q < 0.0)
            /*  Линия вне границы отсечения. */
            returnValue = false;
      return (returnValue);
    }
    void lineClipLiangBarsk (wcPt2D winMin, wcPt2D winMax,
                                       wcPt2D p1, wcPt2D p2)
    {
    GLfloat u1 = 0.0, u2 = 1.0, dx = p2.getx ( ) - p1.getx ( ), dy;

      if (clipTest (-dx, p1.getx ( ) - winMin.getx ( ), &u1, &u2))
        if (clipTest (dx, winMax.getx ( ) - p1.getx ( ), &u1, &u2))
        {
         dy = p2.gety ( ) - p1.gety ( );
         if (clipTest (-dy, p1.gety () - winMin.gety (), &u1, &u2))
          if (clipTest (dy, winMax.gety () - p1.gety (), &u1, &u2))
          {
            if (u2 < 1.0) {
              p2.setCoords (p1.getx ( ) + u2 * dx, p1.gety ( ) +
                                                        u2 * dy);
            }
            if (u1 > 0.0) {
              p1.setCoords (p1.getx ( ) + u1 * dx, p1.gety ( ) +
                                                        u1 * dy);
            }
            lineBres (round (p1.getx ( )), round (p1.gety ( )),
                        round (p2.getx ( )), round (p2.gety()));
          }
        }
}
```

Вообще, алгоритм Лианга–Барски эффективнее алгоритма Коэна–Сазерленда. Каждое обновление параметров u_1 и u_2 требует только одного деления; точки пересечения линии с окном вычисляются только один раз, когда вычисляются конечные значения u_1 и u_2. В то же время, алгоритм Коэна–Сазерленда может последователь-



а) $\mathbf{P}_0$ внутри отсекающего окна б) $\mathbf{P}_0$ в граничной области в) $\mathbf{P}_0$ в угловой области

Рис. 6.16. Три возможных положения конечной точки линии $\mathbf{P}_0$ в алгоритме НЛН

но вычислять точки пересечения вдоль траектории прямой, даже если сам отрезок полностью находится за отсекающим окном. Наконец, каждый этап вычисления точки пересечения по Кохену–Сазерленду требует деления и умножения. В завершение отметим, что алгоритм Лианга–Барски можно расширить на отсечение трехмерных линий (глава 7).

ОТСЕЧЕНИЕ ЛИНИИ НИКОЛЛА–ЛИ–НИКОЛЛА

В алгоритме Николла–Ли–Николла (Nicholl–Lee–Nicholl — NLN, НЛН) создается больше областей вокруг окна отсечения, и за счет этого не нужно несколько раз проводить расчет точек пересечения линии с окном. В методе Коэна–Сазерленда, например, можно вначале вычислить несколько точек пересечения вдоль прямой, содержащей отрезок, перед тем, как будет найдена точка пересечения с собственно отсекающим прямоугольником, или пока линия не будет целиком отброшена. В алгоритме НЛН подобных дополнительных расчетов нет, для чего перед вычислением точки пересечения выполняется больше проверок областей. По сравнению с алгоритмами Коэна–Сазерленда и Лианга–Барски алгоритм НЛН имеет меньше операций сравнения и деления. За эти преимущества нужно платить тем, что алгоритм НЛН применим только к двухмерному отсечению, тогда как два других легко расширяются на трехмерные сцены.

Изначальную проверку, выполняемую, чтобы определить, находится отрезок целиком внутри или целиком снаружи окна, можно провести с использованием кодов областей, как в предыдущих двух алгоритмах. Если тривиальное принятие или отклонение отрезка невозможно, алгоритм НЛН вводит дополнительные области отсечения.

Для отрезка с концами $\mathbf{P}_0$ и $\mathbf{P}_{\text{end}}$ вначале определяется положение точки $\mathbf{P}_0$ в восьми возможных областях относительно отсекающего окна. Из показанных на рис. 6.16 областей нужно рассмотреть только три. Если точка $\mathbf{P}_0$ лежит в одной из шести остальных областей, ее можно переместить в одну из трех областей, показанных на рис. 6.16, используя преобразование симметрии. Например, область непосредственно над отсекающим окном можно преобразовать в область слева от окна, используя отражение относительно линии $y = -x$, или же можно использовать поворот на 90° против часовой стрелки.

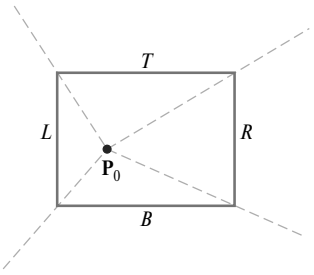


Рис. 6.17. Четыре области, используемые в алгоритме НЛН, когда точка $\mathbf{P}_0$ находится внутри отсекающего окна, а $\mathbf{P}_{\text{end}}$ — вне

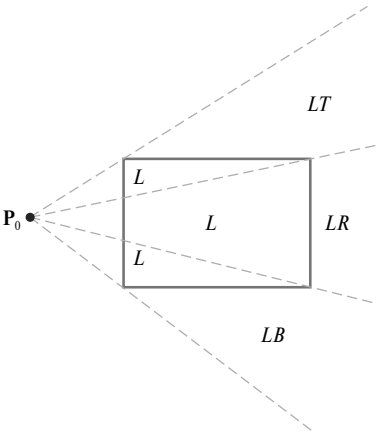


Рис. 6.18. Четыре отсекающие области, используемые в алгоритме НЛН, когда точка $\mathbf{P}_0$ лежит непосредственно слева от отсекающего окна

Предполагая, что $\mathbf{P}_0$ и $\mathbf{P}_{\text{end}}$ не лежат одновременного внутри отсекающего окна, определим положение $\mathbf{P}_{\text{end}}$ относительно $\mathbf{P}_0$. Чтобы это сделать, создадим несколько новых областей на плоскости в зависимости от положения точки $\mathbf{P}_0$. Границы новых областей — лучи, начинающиеся в точке $\mathbf{P}_0$ и проходящие через углы отсекающего окна. Если $\mathbf{P}_0$ находится внутри отсекающего окна, задаются четыре области, показанные на рис. 6.17. Затем в зависимости от того, какая область (L, T, R или B) содержит точку $\mathbf{P}_{\text{end}}$, точка пересечения с линией сравнивается с соответствующей границей окна.

Если $\mathbf{P}_0$ лежит в области слева от окна, задаются четыре области, помеченные на рис. 6.18 L, LT, LR и LB. Данные четыре области определяют единственную сторону отсекающего окна относительно положения $\mathbf{P}_{\text{end}}$. Например, если $\mathbf{P}_{\text{end}}$ находится в любой из трех областей, помеченных L, линия отсекается на левой границе окна, и записывается отрезок от этой точки пересечения до $\mathbf{P}_{\text{end}}$. Если $\mathbf{P}_{\text{end}}$ лежит в области LT, записывается отрезок от левой до верхней границы окна. Подобная обработка выполняется для областей LR и LB. Однако, если $\mathbf{P}_{\text{end}}$ не принадлежит ни одной из областей L, LT, LR и LB, отсекается вся линия.

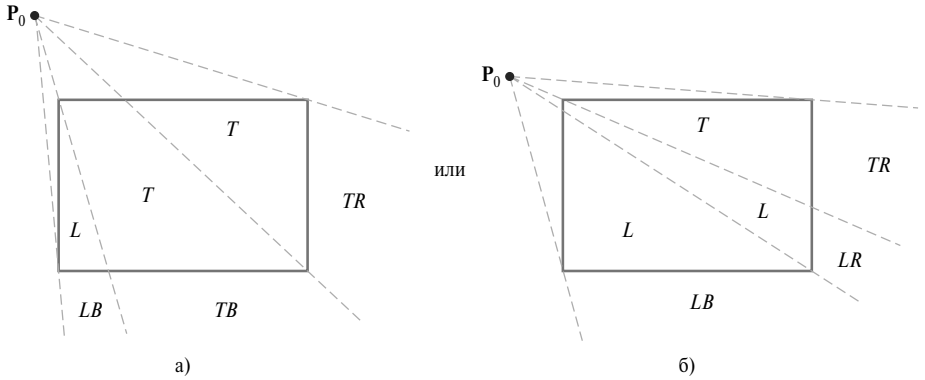


Рис. 6.19. Два возможных набора отсекающих областей, используемых в алгоритме НЛН, когда точка $\mathbf{P}_0$ находится выше и слева от отсекающего окна

В третьем случае, когда $\mathbf{P}_0$ лежит слева сверху отсекающего окна, используются области, показанные на рис. 6.19. В этом случае возможны два показанных на рисунке варианта в зависимости от положения точки $\mathbf{P}_0$ в левом верхнем углу отсекающего окна. Если $\mathbf{P}_0$ ближе к левой границе отсечения, используются области, изображенные на рис. 6.19, *а*. В противном случае, когда $\mathbf{P}_0$ ближе к верхней границе отсечения, используются области, приведенные на рис. 6.19, *б*. Если точка $\mathbf{P}_{\text{end}}$ находится в одной из областей T, L, TR, TB, LR или LB, это условие однозначно определяет границу отсекающего окна для расчета точки пересечения. В противном случае отбрасывается вся линия.

Чтобы определить область, в которой расположена точка $\mathbf{P}_{\text{end}}$, сравниваются тангенсы углов наклона отрезка с тангенсами углов наклона границ областей НЛН. Например, если $\mathbf{P}_0$ находится слева от отсекающего окна (рис. 6.18), тогда $\mathbf{P}_{\text{end}}$ принадлежит области LT, если

$$\overline{\mathbf{P}_0\mathbf{P}_{TR}} < \overline{\mathbf{P}_0\mathbf{P}_{\text{end}}} < \overline{\mathbf{P}_0\mathbf{P}_{TL}} \tag{6.21}$$

или

$$\frac{y_T - y_0}{x_R - x_0} < \frac{y_{\text{end}} - y_0}{x_{\text{end}} - x_0} < \frac{y_T - y_0}{x_L - x_0}. \tag{6.22}$$

Вся линия отсекается, если

$$(y_T - y_0)(x_{\text{end}} - x_0) < (x_L - x_0)(y_{\text{end}} - y_0). \tag{6.23}$$

Расчет разности координат и произведений, используемых в проверках тангенсов углов наклона, записывается, а затем применяется в расчете точек пересечения. Из

параметрических уравнений

$$x = x_0 + (x_{\text{end}} - x_0)u,$$
$$y = y_0 + (y_{\text{end}} - y_0)u$$

вычисляется координата x точки пересечения с левой границей окна $x = x_L$, где $u = (x_L - x_0)/(x_{\text{end}} - x_0)$, так что координата y точки пересечения равна

$$y = y_0 + \frac{y_{\text{end}} - y_0}{x_{\text{end}} - x_0}(x_L - x_0). \tag{6.24}$$

Координата y точки пересечения с верхней границей равна $y = y_T$, и $u = (y_T - y_0)/(y_{\text{end}} - y_0)$, где

$$x = x_0 + \frac{x_{\text{end}} - x_0}{y_{\text{end}} - y_0}(y_T - y_0). \tag{6.25}$$

ОТСЕЧЕНИЕ ЛИНИЙ С ИСПОЛЬЗОВАНИЕМ НЕПРЯМОУГОЛЬНЫХ МНОГОУГОЛЬНЫХ ОКОН

В некоторых приложениях может требоваться отсечь линии многоугольниками произвольной формы. Такие методы, основанные на параметрических уравнениях прямой, как алгоритмы Кируса–Бека или Лианга–Барски, можно легко расширить на отсечение линий выпуклыми многоугольниками. Для этого в алгоритм включаются параметрические уравнения границ отсекающих областей. Предварительный отсев участков линий можно выполнить, обработав линии с указанными координатными границами отсекающего многоугольника.

Для отсекающих областей в форме вогнутых многоугольников данные параметрические процедуры отсечения также применимы, если вначале расщепить вогнутый многоугольник на набор выпуклых, используя один из методов, описанных в разделе 3.15. Другой подход — просто добавить одну или несколько дополнительных сторон к вогнутой отсекающей области, так что она превращается в выпуклую многоугольную форму. Затем, используя видоизмененные выпуклые многоугольные компоненты, можно применить ряд операций отсечения, как показано на рис. 6.20. Итак, нужно знать, как будет выглядеть отрезок $\overline{\mathbf{P}_1\mathbf{P}_2}$ на рис. 6.20, *а* после отсечения вогнутым окном с вершинами $\mathbf{V}_1$, $\mathbf{V}_2$, $\mathbf{V}_3$, $\mathbf{V}_4$ и $\mathbf{V}_5$. В этом случае получается две вогнутые отсекающие области за счет добавления отрезка между $\mathbf{V}_4$ и $\mathbf{V}_1$. Затем линия отсекается в два этапа: 1) линия $\overline{\mathbf{P}_1\mathbf{P}_2}$ отсекается выпуклым многоугольником с вершинами $\mathbf{V}_1$, $\mathbf{V}_2$, $\mathbf{V}_3$ и $\mathbf{V}_4$, и получается обрезанный сегмент $\overline{\mathbf{P}'_1\mathbf{P}'_2}$ (рис. 6.20, *б*); 2) внутренний отрезок $\overline{\mathbf{P}'_1\mathbf{P}'_2}$ отсекается с использованием выпуклого многоугольника с вершинами $\mathbf{V}_1$, $\mathbf{V}_5$ и $\mathbf{V}_4$ (рис. 6.20, *в*), и получается конечный извлеченный отрезок $\overline{\mathbf{P}''_1\mathbf{P}'_2}$.

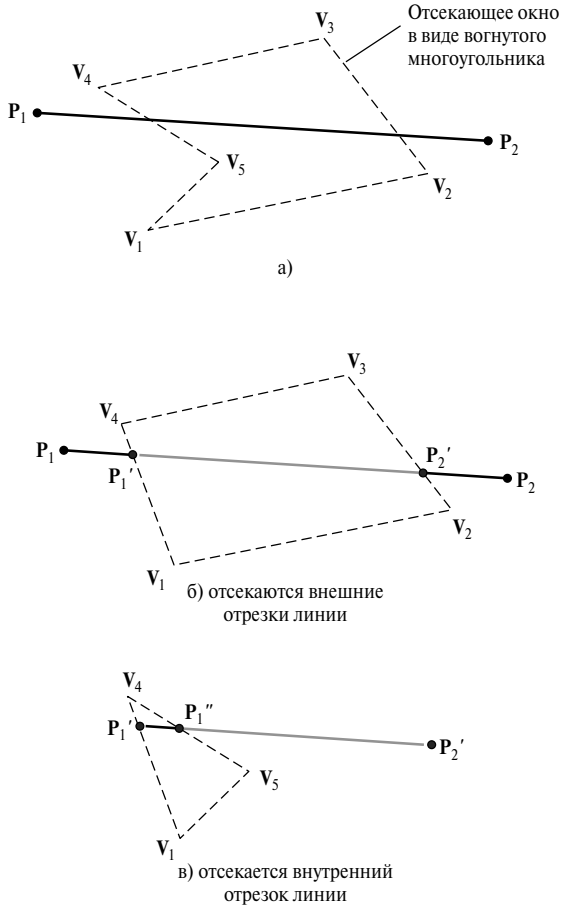
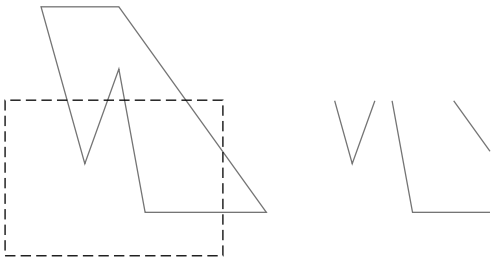


Рис. 6.20. Отсекающее окно в виде вогнутого многоугольника (панель *а*) с вершинами ($\mathbf{V}_1$, $\mathbf{V}_2$, $\mathbf{V}_3$, $\mathbf{V}_4$, $\mathbf{V}_5$) преобразуется в выпуклый многоугольник ($\mathbf{V}_1$, $\mathbf{V}_2$, $\mathbf{V}_3$, $\mathbf{V}_4$) (панель *б*). Внешние сегменты линии $\overline{\mathbf{P}_1\mathbf{P}_2}$ затем отрезаются с использованием данного выпуклого отсекающего окна. Получающийся в результате отрезок $\overline{\mathbf{P}_1'\mathbf{P}_2'}$ затем обрабатывается треугольником ($\mathbf{V}_1$, $\mathbf{V}_5$, $\mathbf{V}_4$) (панель *в*), в результате чего отсекается внутренний участок прямой $\overline{\mathbf{P}_1'\mathbf{P}_1''}$, и получается окончательная линия $\overline{\mathbf{P}_1''\mathbf{P}_2'}$

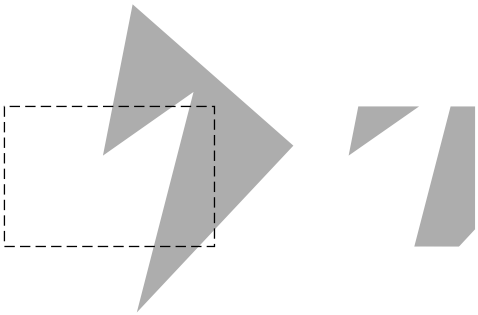
ОТСЕЧЕНИЕ ЛИНИЙ С ИСПОЛЬЗОВАНИЕМ ОКОН С НЕЛИНЕЙНЫМИ ГРАНИЦАМИ

Отметим, что также можно использовать окружности или другие отсекающие области с криволинейными границами, но это требует больше обработки, поскольку расчет точек пересечения включает решение нелинейных уравнений. На первом этапе линии можно укоротить, используя граничный прямоугольник (координатные границы) криволинейной отсекающей области. Линии, расположенные вне координатных границ, удаляются. Чтобы определить линии, которые находятся внутри окружности, например, можно вычислить расстояние конечных точек линии от центра окружности. Если квадрат этого расстояния для конечных точек линии меньше или равен квадрату радиуса, можно извлечь всю линию. Затем нужно обработать оставшиеся линии, и для этого вычисляются точки пересечения, что иногда требует решения систем нелинейных уравнений.



а) до отсечения б) после отсечения

Рис. 6.21. Применение алгоритма отсечения линии к отрезку границы многоугольника (панель *а*) дает несвязанный набор линий (панель *б*)



а) до отсечения б) после отсечения

Рис. 6.22. Изображение правильно извлеченной многоугольной закрашенной области

6.8. ОТСЕЧЕНИЕ МНОГОУГОЛЬНОЙ ЗАКРАШЕННОЙ ОБЛАСТИ

Графические пакеты обычно поддерживают только закрашенные области, являющиеся многоугольниками, причем часто — только выпуклыми. Чтобы обрезать многоугольную закрашенную область, нельзя прямо применить метод отсечения линии к отдельным сторонам многоугольника, поскольку данный подход в общем случае не даст замкнутой ломаной линии. Вместо этого алгоритм отсечения линии часто дает непересекающийся набор линий, причем без указания полной информации о том, как можно сформировать замкнутую границу вокруг извлеченной закрашенной области. На рис. 6.21 иллюстрируется возможный выход процедуры отсечения линий, примененной к сторонам многоугольной закрашенной области. Итак, все, что требуется, — процедура, которая выдаст одну или несколько замкнутых ломаных линий-границ извлеченной закрашенной области, чтобы многоугольники можно было преобразовать в стандарт развертки и заполнить внутренние части заданным цветом или узором, как показано на рис. 6.22.

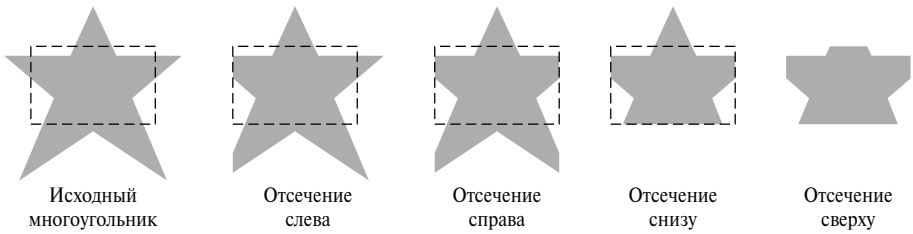


Рис. 6.23. Этапы обработки многоугольной закрашенной области при отсечении прямоугольной областью

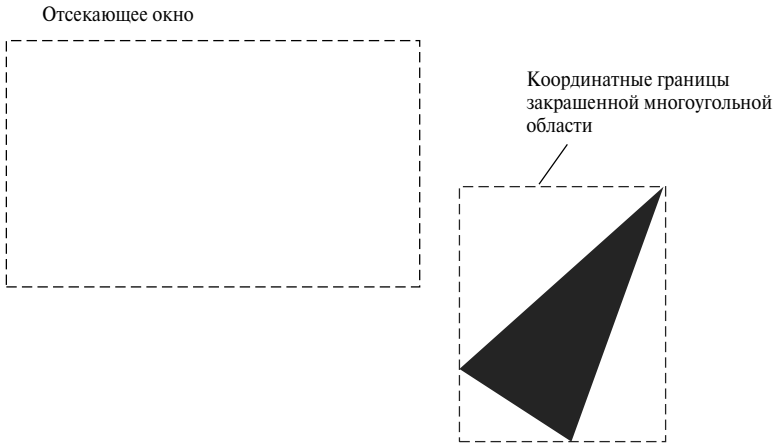


Рис. 6.24. Многоугольная закрашенная область, координатные границы которой находятся за правой границей отсечения

Многоугольную закрашенную область можно извлечь из отсекающего окна, используя тот же общий подход, что и в отсечении линий. Отрезок определяется двумя его концами, и эти точки обрабатываются процедурой отсечения линий, для чего строится новый набор конечных точек, обрезанных на всех границах отсекающего окна. Таким образом, требуется следить, чтобы закрашенная область была единым объектом при обработке на этапах отсечения. Следовательно, для многоугольной закрашенной области можно определять новую форму при обработке каждой стороной отсекающего окна, как показано на рис. 6.23. Разумеется, внутренняя часть многоугольника не будет закрашена, пока не определится окончательная отсеченная граница.

В приведенных выше схемах первая проверка отрезка определяла, можно ли его полностью извлечь или полностью отбросить. Ту же процедуру можно выполнить и для многоугольной закрашенной области, проверив ее координатные границы. Если минимальная и максимальная координаты закрашенной области находятся внутри всех четырех линий отсечения, закрашенная область извлекается для дальнейшей обработки. Если все данные координатные границы находятся вне какой-то границы отсекающего окна, многоугольник исключается из описания сцены (рис. 6.24).

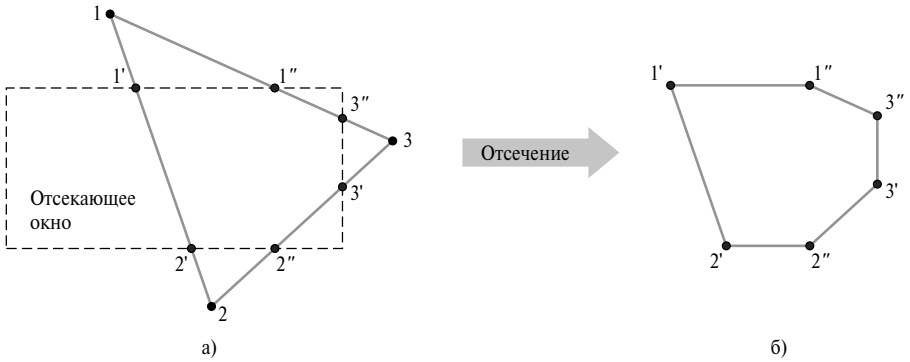


Рис. 6.25. Выпуклая многоугольная закрашенная область (панель *а*), определенная списком вершин {1, 2, 3}, отсекается до получения закрашенной области (панель *б*), определяемой выходным списком вершин $\{1', 2', 2'', 3', 3'', 1''\}$

Если закрашенную область нельзя однозначно отнести к полностью внутренним или полностью внешним по отношению к отсекающему окну, необходимо определить точки пересечения с линиями отсечения. Один метод реализации отрезания заключается в создании нового списка вершин на каждой линии отсечения, а затем передаче этого нового списка вершин следующей процедуре отсечения по линии. Выход конечного этапа отсечения — список вершин извлеченного многоугольника (рис. 6.25). При отсечении вогнутым многоугольником данный базовый подход потребуется несколько модифицировать, так что генерироваться будет несколько списков вершин.

ОТСЕЧЕНИЕ МНОГОУГОЛЬНИКАМИ САЗЕРЛЕНДА–ХОДГМАНА

Эффективный метод извлечения закрашенного выпуклого многоугольника, разработанный Сазерлендом и Ходгманом (Hodgman), заключается в следующем: так пропускать вершины многоугольника по всем этапам отсечения, чтобы отдельную отсеченную вершину можно было сразу передавать на следующий этап. Это позволяет не создавать на каждом этапе отсечения выходной набор вершин, а также реализовать процедуры отсечения по границам параллельно. Окончательный выход — это список вершин, описывающих стороны извлеченной многоугольной закрашенной области.

Поскольку алгоритм Сазерленда–Ходгмана дает только один список выходных вершин, он не может корректно сгенерировать два выходных многоугольника, показанных на рис. 6.22, *б*, полученных при отсечении вогнутого многоугольника, приведенного на рис. 6.22, *а*. В то же время, алгоритм можно изменить, чтобы он давал несколько выходных списков вершин и позволял обрабатывать вогнутые многоугольники общей формы. Стандартный алгоритм Сазерленда–Ходгмана может обрабатывать вогнутые многоугольники, если получающуюся в результате область можно описать одним списком вершин.

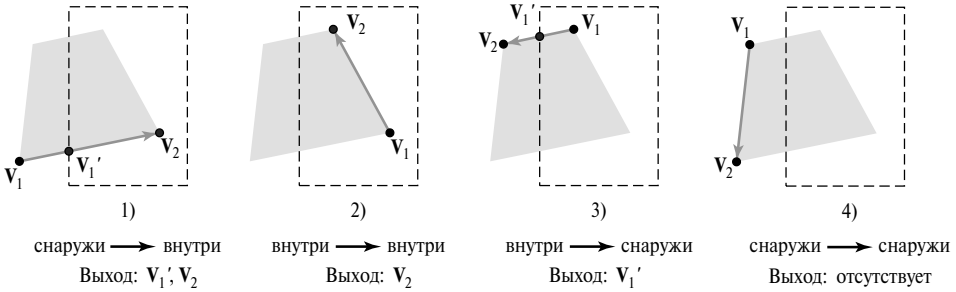


Рис. 6.26. Четыре возможных выхода процедуры отсечения левой границей в зависимости от положения пары конечных точек относительно левой границы отсекающего окна

Общая схема этого алгоритма состоит в следующем: последовательно проводить пары концов соседних сторон многоугольника по ряду процедур отсечения (отсечение слева, справа, снизу, сверху). Когда процедура отсечения завершает обработку пары вершин, выходные координаты оставшихся вершин (если они есть) передаются следующей процедуре отсечения. Затем первая процедура отсечения обрабатывает следующую пару конечных точек. При подобной схеме отдельные процедуры отсечения по границе могут работать параллельно.

При обработке стороны многоугольника процедурой отсечения по границе нужно рассмотреть четыре возможные ситуации. Первая возможность — первый конец отрезка находится снаружи отсекающий границы, а второй — внутри. Вторая — оба конца находятся внутри. Третья ситуация — первая конечная точка находится внутри, а вторая снаружи границы. Наконец, обе точки могут находиться снаружи границы.

Чтобы облегчить передачу вершин с одного этапа отсечения на другой, выход каждой процедуры отсечения можно сформулировать так, как показано на рис. 6.26. При передаче последовательных пар конечных точек одной из четырех процедур отсечения выход генерируется согласно результатам следующих проверок.

1. Если первая входная вершина находится вне данной границы отсекающего окна, а вторая расположена внутри, две точки — пересечения стороны многоугольника с границей окна и вторая вершина — передаются следующей процедуре отсечения.
2. Если обе входные вершины находятся внутри данной границы отсекающего окна, следующей процедуре передается только вторая вершина.
3. Если первая вершина находится внутри данной границы отсекающего окна, а вторая — снаружи, следующей процедуре передается только точка пересечения стороны многоугольников с границей отсекающего окна.
4. Если обе входные вершины находятся вне данной границы отсекающего окна, ни одна из них не передается следующей процедуре.

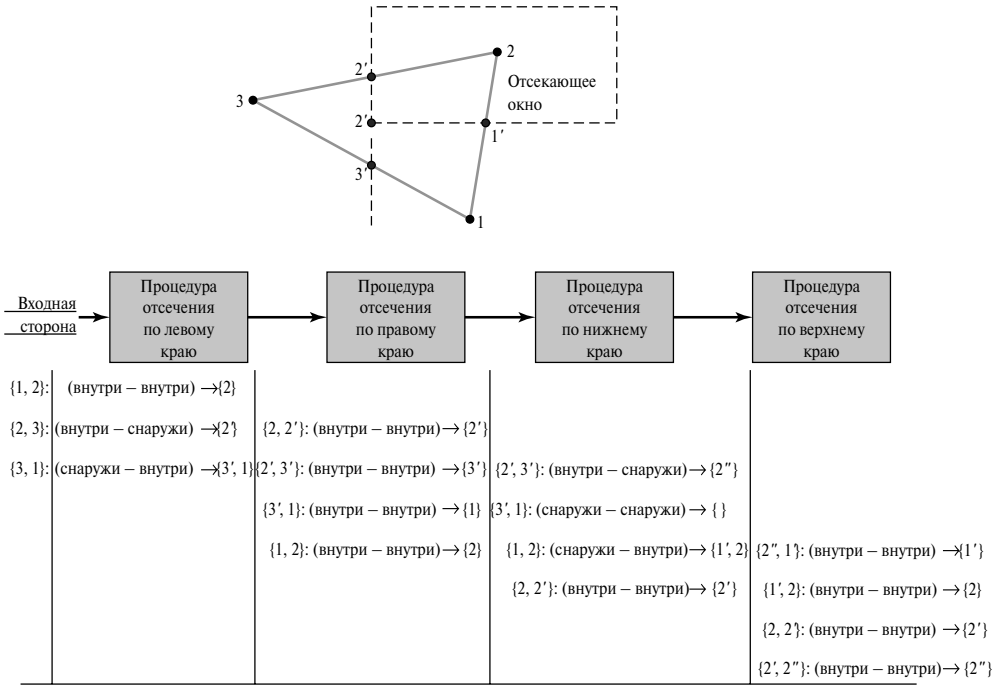


Рис. 6.27. Обработка набора вершин многоугольника $\{1,\ 2,\ 3\}$ процедурами отсечения по границам с использованием алгоритма Сазерленда–Ходгмана. Набор вершин после отсечения — $\{1', 2, 2', 2''\}$

Последняя в цикле процедура отсечения генерирует список вершин, который описывает окончательную обрезанную закрашенную область.

На рис. 6.27 изображен пример алгоритма Сазерленда–Ходгмана для закрашенной области, определенный списком вершин $\{1, 2, 3\}$. Пока процедура отсечения получает пару конечных точек, она определяет соответствующий выход, используя проверки, иллюстрируемые на рис. 6.26. Данные выходы последовательно передаются от процедуры отсечения по левой границе процедурам отсечения по правой, нижней и верхней границам. Выход процедуры отсечения по верхней границе представляет собой набор вершин, определяющих окончательную закрашенную область. В приведенном примере выходной список вершин выглядит как $\{1', 2, 2', 2''\}$.

Последовательная реализация алгоритма отсечения многоугольников Сазерленда–Ходгмана демонстрируется в следующем наборе процедур. Входной набор вершин преобразуется в выходной путем применения процедур отсечения слева, справа, снизу и сверху.

```
typedef enum { Left, Right, Bottom, Top } Boundary;
const GLint nClip = 4;
GLint inside (wcPt2D p, Boundary b, wcPt2D wMin, wcPt2D wMax)
{ switch (b) {
  case Left:   if (p.x < wMin.x) return (false); break;
  case Right:  if (p.x > wMax.x) return (false); break;
  case Bottom: if (p.y < wMin.y) return (false); break;
  case Top:    if (p.y > wMax.y) return (false); break;  }
  return (true);
}
GLint cross (wcPt2D p1, wcPt2D p2, Boundary winEdge,
                                wcPt2D wMin, wcPt2D wMax)
{
  if (inside (p1, winEdge, wMin, wMax) == inside (p2, winEdge,
        wMin, wMax))
    return (false);
  else return (true);
}
wcPt2D intersect (wcPt2D p1, wcPt2D p2, Boundary winEdge,
                                     wcPt2D wMin, wcPt2D wMax)
{
  wcPt2D iPt;
  GLfloat m;

  if (p1.x != p2.x) m = (p1.y - p2.y) / (p1.x - p2.x);
  switch (winEdge) {
  case Left:
    iPt.x = wMin.x;
    iPt.y = p2.y + (wMin.x - p2.x) * m;
    break;
  case Right:
    iPt.x = wMax.x;
    iPt.y = p2.y + (wMax.x - p2.x) * m;
    break;
  case Bottom:
    iPt.y = wMin.y;
    if (p1.x != p2.x) iPt.x = p2.x + (wMin.y - p2.y) / m;
    else iPt.x = p2.x;
    break;
  case Top:
    iPt.y = wMax.y;
    if (p1.x != p2.x) iPt.x = p2.x + (wMax.y - p2.y) / m;
    else iPt.x = p2.x;
    break;
  }
  return (iPt);
}
```

```
void clipPoint (wcPt2D p, Boundary winEdge, wcPt2D wMin,
                wcPt2D wMax, wcPt2D * pOut, int * cnt,
                wcPt2D * first[], wcPt2D * s)
{
  wcPt2D iPt;

  /* Если для данной границы отсечения не существует
   * предыдущей точки, записывается текущая точка.
   */
  if (!first[winEdge])
    first[winEdge] = &p;
  else

/* Предыдущая точка существует. Если отрезок, соединяющий p
 * и предыдущую точку, пересекает данную границу отсечения,
 * находится точка пересечения. Если еще есть границы,
 * отсечение по которым не выполнялось, провести отсечение
 * по ним. Если границ отсечения больше нет, точки пересечения
 * добавляются в выходной список.
 */
    if (cross (p, s[winEdge], winEdge, wMin, wMax)) {
      iPt = intersect (p, s[winEdge], winEdge, wMin, wMax);
      if (winEdge < Top)
        clipPoint (iPt, b+1, wMin, wMax, pOut, cnt, first, s);
      else {
        pOut[*cnt] = iPt;  (*cnt)++;
      }
    }

  /* Для данной границы отсечения записать p как новую
   * точку.
   */
  s[winEdge] = p;

  /* Если точка внутренняя, перейти к следующей необработанной
   * границе (если такие остались).
   */
  if (inside (p, winEdge, wMin, wMax))
   if (winEdge < Top)
   clipPoint (p, winEdge + 1, wMin, wMax, pOut, cnt, first, s);
   else {
      pOut[*cnt] = p;  (*cnt)++;
    }
}
```

```
void closeClip (wcPt2D wMin, wcPt2D wMax, wcPt2D * pOut,
                GLint * cnt, wcPt2D * first [ ], wcPt2D * s)
{
  wcPt2D pt;
  Boundary winEdge;

  for (winEdge = Left; winEdge <= Top; winEdge++) {
    if (cross (s[winEdge], *first[winEdge], winEdge,
        wMin, wMax)) { pt = intersect (s[winEdge],
        *first[winEdge], winEdge, wMin, wMax);
      if (winEdge < Top)
        clipPoint (pt, winEdge + 1, wMin, wMax, pOut, cnt,
                                                  first, s);
      else {
        pOut[*cnt] = pt;  (*cnt)++;
      }
   }
  }
}
GLint polygonClipSuthHodg (wcPt2D wMin, wcPt2D wMax, GLint n,
                           wcPt2D * pIn, wcPt2D * pOut)
{
  /* Параметр "first" содержит указатель на первую точку,
   * обработанную для границы; "s" содержит последнюю
   * точку, обработанную с данной границей.
   */
  wcPt2D * first[nClip] =  0, 0, 0, 0 , s[nClip];
  GLint k, cnt = 0;

  for (k = 0; k < n; k++)
    clipPoint (pIn[k], Left, wMin, wMax, pOut, &cnt, first, s);
  closeClip (wMin, wMax, pOut, &cnt, first, s);
  return (cnt);
}
```

Если с помощью алгоритма Сазерленда–Ходгмана обрабатывается вогнутый многоугольник, на экран могут выводиться посторонние линии. Пример такой ситуации приведен на рис. 6.28. Это происходит, когда обработанный многоугольник должен иметь несколько отдельных участков, но, поскольку существует только один выходной список вершин, последняя вершина списка всегда соединяется с первой.

Для корректного отсечения вогнутых многоугольников можно предпринять некоторые действия. Во-первых, вогнутый многоугольник можно расщепить на несколько выпуклых (раздел 3.15) и обработать каждый полученный многоугольник отдельно с использованием алгоритма Сазерленда–Ходгмана. Во-вторых, можно модифицировать этот алгоритм, чтобы конечный набор вершин проверялся на наличие нескольких точек пересечения вдоль границы отсекающего окна. Если обнаружено более двух

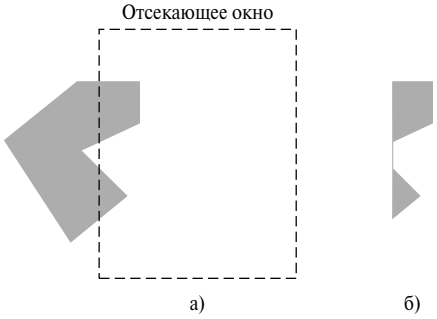


Рис. 6.28. Отсечение вогнутого многоугольника (панель *а*) с использованием алгоритма Сазерленда–Ходгмана дает две связанные области (панель *б*)

вершин вдоль границы отсечения, данный список можно разделить на несколько списков, верно идентифицирующих участки обрезанной закрашенной области. Это может требовать досконального анализа, выявляющего, точки вдоль границы отсечения образуют пары или являются отдельными вершинами конечного многоугольника. В-третьих, можно использовать более общую процедуру отсечения многоугольников, разработанную для корректной обработки вогнутых многоугольников.

АЛГОРИТМ ОТСЕЧЕНИЯ МНОГОУГОЛЬНИКОВ УЭЙЛЕРА–АЗЕРТОНА (WEILER–ATHERTON)

Данный алгоритм — это общий подход к отсечению многоугольников, который можно использовать для отсечения закрашенной области выпуклого или вогнутого многоугольника. Более того, этот метод разрабатывался как средство идентификации видимых поверхностей трехмерной сцены. Следовательно, его также можно использовать для отсечения любой многоугольной области с помощью окна произвольной многоугольной формы.

Вместо того чтобы просто отсечь стороны закрашенной области, как в методе Сазерленда–Ходгмана, алгоритм Уэйлера–Азертона обрабатывает периметр закрашенной области и находит границы, замыкающие отсеченную закрашенную область. Таким образом, несколько закрашенных областей (см. рис. 6.28, *б*) можно идентифицировать и отобразить как отдельные, несвязанные многоугольники. Чтобы найти стороны отсеченной многоугольной области, проходится путь (по часовой стрелке или против нее) вокруг закрашенной области, который обходит границу отсекающего окна там, где сторона многоугольника выходит из данного периметра. Направление обхода границы отсекающего окна совпадает с направлением обработки сторон многоугольника.

Обычно можно определить, каким является направление обработки — по часовой стрелке или против нее — упорядоченного списка вершин, определяющего многоугольную закрашенную область. В большинстве случаев список вершин задается против часовой стрелки, как в правиле определения передней грани многоугольника. Сле-

довательно, векторное произведение векторов двух последовательных сторон, формирующих выпуклый угол, определяет направление вектора нормали — направление от задней грани многоугольника к передней. Если упорядочение вершин неизвестно, вектор нормали можно либо вычислить, либо воспользоваться любым методом, рассмотренным в разделе 3.15 для определения внутренней части закрашенной области. Затем, если последовательно обрабатывать края так, чтобы внутренняя часть многоугольника всегда находилась слева, получим обход против часовой стрелки. В противном случае, оставляя внутреннюю часть справа, получим обход по часовой стрелке.

При обходе вершин многоугольника против часовой стрелки применяются следующие процедуры Уэйлера–Азертона.

1. Обрабатывать стороны многоугольной закрашенной области в направлении против часовой стрелки, пока для одной из границ отсечения не встретится пара "внутренняя-внешняя вершина"; т.е. первая вершина стороны многоугольника находится внутри обрабатываемой области, а вторая — вне ее.
2. Проходить границы окна в направлении против часовой стрелки от выходной точки пересечения до другой точки пересечения с многоугольником. Если это — ранее обработанная точка, следует перейти к следующему этапу. Если это — новая точка пересечения, обработка сторон многоугольника продолжается в порядке против часовой стрелки, пока не встретится ранее обработанная вершина.
3. Для данного участка обрабатываемой закрашенной области сформировать список вершин.
4. Вернуться к точке пересечения-выхода и продолжать обработку сторон многоугольника против часовой стрелки.

На рис. 6.29 иллюстрируется отсечение вогнутого многоугольника по Уэйлеру–Азертону с помощью стандартного окна отсечения для обхода сторон многоугольника против часовой стрелки. При обходе сторон по часовой стрелке обход окна отсечения удобно выполнять аналогично.

Если начать с вершины 1 на рис. 6.29, *а*, то против часовой стрелки следующей обрабатываемой вершиной будет вершина 2. Следовательно, данная сторона существует на верхней границе отсекающего окна. Эта точка $1'$ вычисляется, а затем выполняется левый поворот для обработки границ окна против часовой стрелки. Обрабатывая верхнюю границу отсекающего окна, мы не пересекаем сторону многоугольника, пока не дойдем до левой границы окна. Таким образом, эта вершина помечается $1''$, и мы следуем по левой границе до точки пересечения $1'''$. Затем мы следуем по этой стороне против часовой стрелки, возвращаясь к вершине 1. Этим мы завершаем обход границ окна, и список вершин $\{1, 1', 1'', 1'''\}$ определяет обработанную область исходной закрашенной области. Обработка сторон многоугольника затем возобновляется в точке $1'$. Сторона, определенная точками 2 и 3, выходит из левой границы, но точки 2 и $2'$ находятся над верхней стороной отсекающего окна, а точки

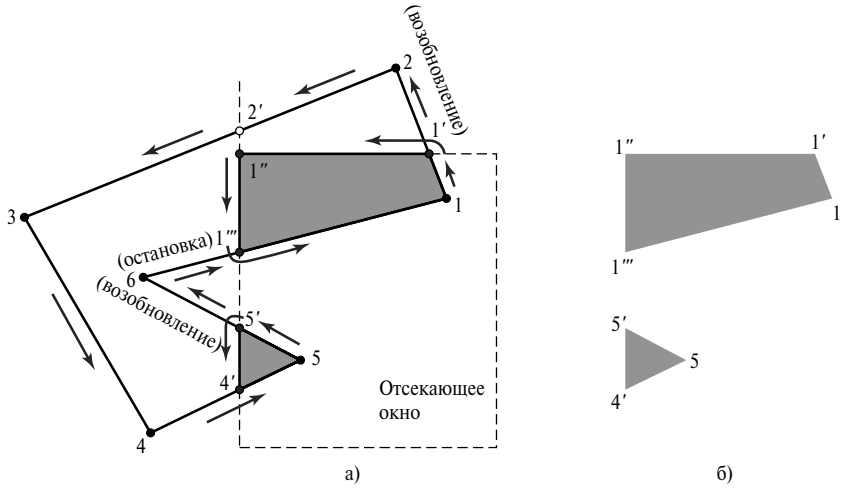


Рис. 6.29. Вогнутый многоугольник (панель *а*), определенный списком вершин $\{1, 2, 3, 4, 5, 6\}$, при вырезании с использованием алгоритма Уэйлера–Азертона дает два списка $\{1, 1', 1'', 1'''\}$ и $\{4', 5, 5'\}$, представляющих два отдельных закрашенных многоугольника (панель *б*)

$2'$ и 3 — слева от отсекаемой области. Кроме того, сторона с конечными точками 3 и 4 расположена снаружи левой границы отсечения. Однако следующая сторона (от точки 4 до точки 5) снова входит в извлекаемую область, и нужно обработать точку пересечения $4'$. Кроме того, сторона с концами 5 и 6 выходит из окна в точке $5'$, поэтому мы обходим снизу левую границу отсечения и получаем замкнутый список вершин $\{4', 5, 5'\}$. Мы продолжаем обработку сторон многоугольника в точке $5'$, что возвращает нас к ранее обработанной точке $1'''$. В этой точке все вершины и стороны многоугольника обработаны, так что вся закрашенная область полностью извлечена.

ОБРАБОТКА МНОГОУГОЛЬНИКОВ С ИСПОЛЬЗОВАНИЕМ НЕПРЯМОУГОЛЬНЫХ ОТСЕКАЮЩИХ ОКОН

Алгоритм Лианга–Барски и другие параметрические методы отсечения линий особенно хорошо подходят для обработки многоугольных закрашенных областей с помощью отсекающих окон в форме выпуклых многоугольников. В этом подходе используется параметрическое представление сторон закрашенной области и отсекающего окна, и оба многоугольника представляются списком вершин. Вначале сравниваются положения прямоугольников, ограничивающих закрашенную область и отсекающий многоугольник. Если закрашенную область невозможно назвать полностью внешней по отношению к отсекающему многоугольнику, можно использовать проверки “внутри-снаружи” для обработки параметрических уравнений сторон. После проверки всех областей решаются системы параметрических уравнений линий, и определяются точки пересечения с окном.

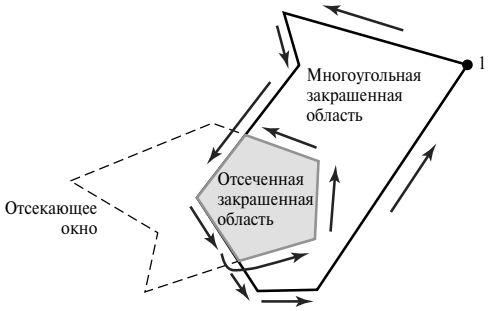


Рис. 6.30. Обработка многоугольной закрашенной области отсекающим окном в форме вогнутого многоугольника с использованием алгоритма Уэйлера–Азертона

Любую многоугольную область можно также обработать любым отсекающим окном в форме многоугольника (выпуклого или вогнутого), как показано на рис. 6.30, используя схему обхода сторон (алгоритм Уэйлера–Азертона). В этом случае необходимо поддерживать для отсекающего окна и многоугольной области списки вершин, упорядоченные против часовой стрелки или по ней. Кроме того, нужно применить проверки “внутри-снаружи”, чтобы определить, находится вершина закрашенной области внутри или снаружи определенной границы отсекающего окна. Как и в предыдущем примере, границы окна прослеживаются, и определяется, где сторона закрашенной области выходит из отсекающей границы. Данный метод отсечения можно также использовать, если закрашенная область или отсекающее окно содержит отверстия, определенные многоугольными границами. Кроме того, данный базовый подход в конструктивной блочной геометрии используется для определения объединения, пересечения или разности двух многоугольников. Фактически вычисление отрезанной области закрашенной фигуры равнозначно определению пересечения двух плоских областей.

ОБРАБОТКА МНОГОУГОЛЬНИКА С ИСПОЛЬЗОВАНИЕМ ОТСЕКАЮЩИХ ОКОН С НЕЛИНЕЙНЫМИ ГРАНИЦАМИ

Один метод обработки с помощью отсекающего окна с криволинейными границами заключается в аппроксимации границ прямыми отрезками и использовании одного из алгоритмов отсечения с помощью многоугольного окна. В качестве альтернативы можно использовать те же общие процедуры, что обсуждались для отрезков. Во-первых, можно сравнить координатные границы закрашенной области с координатными границами отсекающего окна. В зависимости от формы отсекающего окна также возможны другие проверки областей, основанные на соображениях симметрии. Для закрашенных областей, которые нельзя однозначно отнести к внутренним или внешним, в конечном итоге потребуется вычислить точки пересечения закрашенной области с окном.

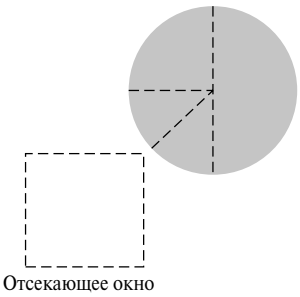


Рис. 6.31. Круговая закрашенная область, демонстрирующая квадрант и октант, находящиеся вне границ отсекающего окна

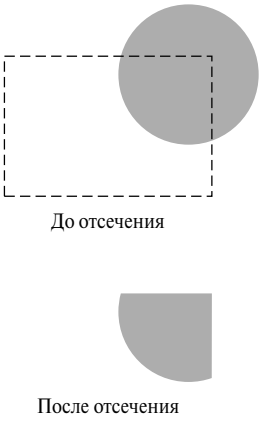


Рис. 6.32. Отсечение круговой закрашенной области

6.9. ОТСЕЧЕНИЕ КРИВЫХ

Области с криволинейными границами можно подвергать отсечению, используя методы, подобные рассмотренным в предыдущих разделах. Если объекты аппроксимируются прямыми отрезками, используются методы отсечения многоугольников. В противном случае процедуры отсечения включают нелинейные уравнения, а следовательно, требуют больше обработки для объектов с нелинейными границами.

Итак, вначале можно сравнить координатные границы объекта с границами отсечения, определив, можно ли тривиально принять или отклонить объект целиком. Если нет, проверяется симметрия объекта, которую, возможно, удастся использовать в первоначальных проверках принятия-отклонения. Например, окружности имеют симметричные квадранты и октанты, так что можно проверять координатные границы этих отдельных секторов окружности. Всю круговую область, показанную на рис. 6.31, нельзя отклонить полностью, просто проверив ее общие координатные границы. Однако половина окружности находится вне правой границы отсечения (или вне верхней границы), левый верхний квадрант расположен над верхней границей отсечения, подобным образом можно устранить оставшиеся два октанта.

Расчет точек пересечения включает подстановку точки на границе отсечения ($xw_{\min}$, $xw_{\max}$, $yw_{\min}$ или $yw_{\max}$) в нелинейное уравнение границы объекта и вычисление значения другой координаты. После того как все точки пересечения будут вычислены, положение определяющих точек объекта можно записать для дальнейшего использования в процедурах закрашивания по строкам развертки. На рис. 6.32 иллюстрируется окружность, обрезаемая прямоугольным окном. В данном примере радиус окружности и конечные точки отсекаемой дуги можно применить для закрашивания вырезанной области.

Подобные процедуры можно применить при обработке криволинейного объекта отсекающим многоугольником. При первом проходе алгоритма можно сравнить граничный прямоугольник объекта с граничным прямоугольником отсекающей области. Если это не позволит полностью записать или удалить объект, далее решается система уравнений кривых, из которой определяются точки пересечения с отсекающей фигурой.

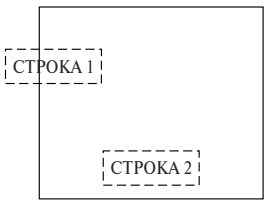
6.10. ОТСЕЧЕНИЕ ТЕКСТА

Существует несколько методов, которые можно использовать для отсечения текста в графическом пакете. В определенном приложении выбор метода отсечения зависит от того, как генерируются символы, и какие требования наложены на отображение строк символов на экране.

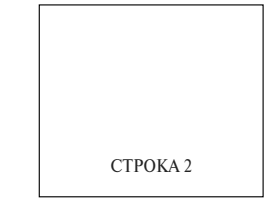
Простейший метод обработки строк символов отсекающим окном заключается в использовании стратегии *все или ничего*, показанной на рис. 6.33. Если все строки находятся внутри отсекающего окна, отображается вся строка. В противном случае все строка удаляется. Данная процедура реализуется путем изучения координатных границ строки текста. Если координатные границы ограничивающего прямоугольника не входят полностью в отсекающее окно, строка отклоняется.

Альтернативой является использование стратегии *отсечения символов по принципу "все или ничего"*. Здесь удаляются только те символы, которые не входят полностью в отсекающее окно (рис. 6.34). В этом случае координатные границы отдельных символов сравниваются с границами окна. Любой символ, не входящий полностью в отсекающее окно, удаляется.

Третий подход к отсечению текста состоит в отсечении компонентов отдельных символов. Это позволяет наиболее точно отразить строки символов после отсечения, но требует больше всего обработки. Символы — это не совсем то же, что линии или многоугольники. Если отдельный символ накладывается на отсекающее окно, отсекаются только части символа, находящиеся вне окна (рис. 6.35). Символы эскизных шрифтов, определенные отрезками, обрабатываются подобным способом с использованием алгоритмов отсечения многоугольников. Обработка растровых символов заключается в сравнении относительных положений отдельных пикселей в точечных изображениях символов с границами отсекающей области.

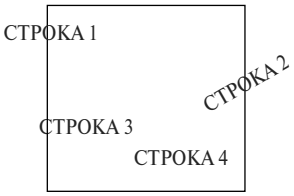


До отсечения



После отсечения

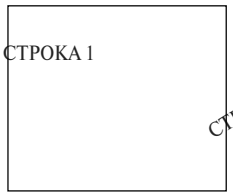
Рис. 6.33. Отсечение текста с использованием координатных границ строки целиком



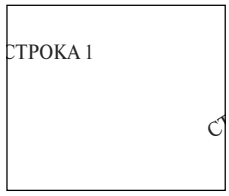
До отсечения

После отсечения

Рис. 6.34. Отсечение текста с использованием прямоугольников, ограничивающих отдельные символы строки



До отсечения



После отсечения

Рис. 6.35. Отсечение текста, выполненное для компонентов отдельных символов

6.11. РЕЗЮМЕ

Двухмерный конвейер преобразования наблюдения — это набор операций, приводящих к выводу на экран изображения во внешних координатах, определенного на плоскости xy. Построенную сцену можно отобразить в систему координат наблюдения, а затем — в нормированную систему координат, где можно применять процедуры отсечения. Наконец, сцена преобразовывается в координаты устройства для вывода на экран. Нормированные координаты можно задать в диапазоне от 0 до 1 или в диапазоне от -1 до 1, и они нужны для того, чтобы графические пакеты были независимы от требований устройств вывода.

Часть сцены для отображения на устройстве вывода выбирается с использованием отсекающего окна, которое можно описать во внешней системе координат или в системе координат наблюдения, определенной относительно внешних координат. Содержимое отсекающего окна передается полю просмотра для отображения на устройства вывода. В некоторых системах поле просмотра задается в нормированных координатах. Другие системы задают поле просмотра в координатах устройства. Обычно отсекающее окно и поле просмотра — это прямоугольники, стороны которых параллельны координатным осям. Объект отображается в поле просмотра так, что он имеет то же относительное положение в поле просмотра, которое у него было в отсекающем окне. Чтобы сохранить относительные пропорции объекта, поле просмотра должно иметь то же самое характеристическое отношение, что и соответствующее

отсекающее окно. Кроме того, для сцены можно задавать любое число отсекающих окон и полей просмотра.

Алгоритмы отсечения обычно реализуются в нормированных координатах, так что все геометрические преобразования и операции наблюдения, независимые от координат устройства, можно объединить в одну матрицу преобразования. Когда поле просмотра задается в координатах устройства, двухмерную сцену можно обрезать с помощью симметричного квадрата, нормированные координаты которого меняются от -1 до 1, а затем передать содержимое нормированного симметричного квадрата в поле просмотра.

Все графические пакеты имеют процедуры для отсечения прямых отрезков и прямоугольных закрашенных областей. Пакеты, содержащие функции задания отдельных точек или текстовых строк, также содержат процедуры отсечения этих графических примитивов. Поскольку расчет точек пересечения является ресурсоемким процессом, разработка улучшенных алгоритмов отсечения является важной областью в сфере компьютерной графики. Кохен и Сазерленд разработали алгоритм отсечения линий, который использует код области для определения положения конечной точки линии относительно границ отсекающего окна. Коды областей конечных точек позволяют быстро определить те линии, которые полностью лежат внутри отсекающего окна, и некоторые линии, которые полностью находятся снаружи. Для оставшихся линий нужно вычислить точки пересечения с границами окна. Лианг и Барский разработали более быстрый алгоритм отсечения линий, в котором отрезки представлены параметрическими уравнениями, подобными алгоритму Кируса–Бека. Данный подход позволяет выполнить больше проверок перед обработкой с целью расчета точек пересечения. Наконец, алгоритм Николла–Ли–Николла еще больше сокращает расчеты точек пересечения за счет использованиия более интенсивной проверки областей на плоскости xy. Параметрические методы отсечения линий легко расширяются на вогнутые отсекающие окна и трехмерные сцены. В то же время, алгоритм Николла–Ли–Николла применим только к двухмерным отрезкам.

Существуют также алгоритмы отсечения прямых отрезков с помощью вогнутых многоугольных окон. Один подход к этой задаче заключается в расщеплении вогнутого многоугольного отсекающего окна на набор выпуклых многоугольников и применении методов параметрического отсечения линий. Другой подход — добавить стороны к вогнутому окну и преобразовать его в выпуклую форму. Затем можно выполнить ряд внутренних и внешних операций отсечения, в ходе которых находится извлекаемый отрезок.

Хотя отсекающие окна с криволинейными границами используются редко, можно разработать соответствующие методы отсечения линий, однако теперь вычисление точек пересечения будет включать решение нелинейных уравнений.

Закрашенная многоугольная область определяется списком вершин, а процедуры отсечения многоугольников должны содержать информацию о том, как извлеченные стороны соединяются при передаче многоугольника по различным этапам обработки. В алгоритме Сазерленда–Ходгмана пары вершин закрашенной области последовательно обрабатываются всеми процедурами отрезания по границе обработки, и информация по данной стороне немедленно передается следующей процедуре отсечения, что позволяет параллельно выполнять четыре процедуры обработки (отсечение слева, справа, снизу и сверху). Данный алгоритм является эффективным методом отсечения закрашенных областей в виде вогнутых многоугольников. Однако, если обрезаемый вогнутый многоугольник содержит несвязанные области, алгоритм Сазерленда–Ходгмана дает посторонние соединяющие отрезки. Расширения процедур параметрического отсечения, таких как метод Лианга–Барски, также могут использоваться для отсечения выпуклых закрашенных областей. Алгоритм Уэйлера–Азертона, в котором применяется обход границы, позволяет корректно обрабатывать как выпуклые, так и вогнутые закрашенные области.

Закрашенные области можно обрабатывать отсекающими выпуклыми окнами, используя расширение схемы параметрического представления прямых. Кроме того, метод Уэйлера–Азертона позволяет обрабатывать любую многоугольную закрашенную область, используя любое отсекающее окно в форме многоугольника. Закрашенные области можно вырезать и окнами с нелинейными границами, используя аппроксимацию многоугольниками или обрабатывая закрашенные области отсекающими окнами с криволинейными границами.

Самым быстрым методом отсечения текста является стратегия “все или ничего”, которая полностью отсекает строку текста, если любая часть строки находится вне границы отсекающего окна. Строку текста можно также обработать, удалив только те символы строки, которые не находятся полностью в отсекающем окне. Наконец, самым точным методом отсечения текста является применение схемы отсечения точек, линий, многоугольников или кривых к отдельным символам строки в зависимости от того, определены символы как точечные изображения или эскизные шрифты.

Хотя OpenGL предназначен для трехмерных приложений, существует двухмерная функция GLU, задающая стандартное прямоугольное отсекающее окно во внешних координатах. В OpenGL координаты отсекающего окна являются параметрами проективного преобразования. Следовательно, вначале необходимо вызвать режим матрицы проектирования. Затем можно задать поле просмотра, используя функции стандартной библиотеки OpenGL, и окно на экране, используя функции GLUT. Существует множество функций GLUT, позволяющих настраивать различные параметры окна на экране. В табл. 6.1 приведено резюме по функциям двухмерного наблюдения OpenGL. Кроме того, в таблице перечислены некоторые функции, связанные с наблюдением.

ТАБЛИЦА 6.1. Резюме по функциям OpenGL двухмерного наблюдения

Функция	*Описание*
`gluOrtho2D`	Задает координаты отсекающего окна как параметры двухмерной ортогональной проекции
`glViewport`	Задает параметры поля просмотра в экранных координатах
`glGetIntegerv`	Использует аргументы `GL_VIEWPORT` и `vpArray` для получения параметров текущего активного поля просмотра
`glutInit`	Инициализирует библиотеку GLUT
`glutInitWindowPosition`	Задает координаты левого верхнего угла окна на экране
`glutInitWindowSize`	Задает ширину и высоту окна на экране
`glutCreateWindow`	Создает окно на экране (которому присваивается целочисленный идентификатор) и задает название окна на экране
`glutInitDisplayMode`	Выбирает параметры, такие как режим буферизации и цвета для окна на экране
`glClearColor`	Задает RGB-цвет фона окна на экране
`glClearIndex`	Задает фоновый цвет окна на экране с использованием режима индексации
`glutDestroyWindow`	Задает число — идентификатор удаляемого окна
`glutSetWindow`	Задает число — идентификатор текущего окна
`glutPositionWindow`	Обновляет положение на экране текущего окна
`glutReshapeWindow`	Обновляет ширину и высоту текущего окна на экране
`glutFullScreen`	Устанавливает размер текущего окна равным размеру экрана
`glutReshapeFunc`	Задает функцию, вызываемую при изменении размера окна
`glutIconifyWindow`	Превращает текущее окно в пиктограмму
`glutSetIconTitle`	Задает метку для пиктограммы окна
`glutSetWindowTitle`	Задает новое название текущего окна

Окончание табл. 6.1

Функция	*Описание*
`glutPopWindow`	Перемещает текущее окно “наверх”; т.е. ставит его перед всеми остальными окнами
`glutPushWindow`	Перемещает текущее окно “вниз”; т.е. располагает его позади всех остальных окон
`glutShowWindow`	Возвращает текущее окно на экран
`glutCreateSubWindow`	Создает окно второго уровня в данном окне
`glutSetCursor`	Определяет форму курсора экрана
`glutDisplayFunc`	Вызывает функцию создания изображения в текущем окне
`glutPostRedisplay`	Обновляет содержимое текущего окна
`glutMainLoop`	Выполняет программу компьютерной графики
`glutIdleFunc`	Задает функцию, выполняемую, когда система не выполняет никаких действий
`glutGet`	Запрашивает систему о заданном параметре состояния

ЛИТЕРАТУРА

Алгоритмы отсечения линий обсуждаются в работах [73, 191, 231, 335]. Методы повышения скорости алгоритма отсечения линий Коэна–Сазерленда представлены в статье [86].
Стандартные методы отсечения многоугольниками описаны в публикациях [190, 343]. Общие методы отсечения многоугольников произвольной формы многоугольными окнами произвольной формы представлены в работах [380, 381].
Операции наблюдения OpenGL обсуждаются в руководстве [400]. Процедуры окон GLUT рассмотрены в книге [174], а дополнительную информацию по GLUT можно найти на Web-сайте `http://reality.sgi.com/opengl/glut3/glut3.html`.

УПРАЖНЕНИЯ

6.1. Напишите процедуру расчета элементов матрицы (6.1) для преобразования двухмерных внешних координат в координаты наблюдения для данного начала системы координат наблюдения $\mathbf{P}_0$ и вектора верха $\mathbf{V}$.

6.2. Выведите матрицу (6.8) перевода содержимого отсекающего окна в поле просмотра через масштабирование окна до размера поля с последующей трансляцией масштабированного окна в положение поля. В качестве опорной точки при масштабировании и трансляции используйте центр отсекающего окна.

6.3. Напишите процедуру расчета элементов матрицы (6.9), переводящей отсекающее окно в симметричный нормированный квадрат.

6.4. Напишите ряд процедур для реализации двухмерного конвейера наблюдения без операций отсечения. Программа должна разрешать построение сцены с помощью преобразований в модельных координатах, задание системы наблюдения и преобразование в симметричный нормированный квадрат. Можно дополнительно реализовать таблицу наблюдения для хранения различных наборов параметров преобразования наблюдения.

6.5. Напишите полную программу с реализацией алгоритма отсечения линий Коэна–Сазерленда.

6.6. Подробно обсудите связь различных проверок и методов расчета параметров пересечения u_1 и u_2 в алгоритме отсечения линий Лианга–Барски.

6.7. Сравните число арифметических операций в алгоритмах Коэна–Сазерленда и Лианга–Барски для нескольких различных ориентаций линии относительно отсекающего окна.

6.8. Напишите полную программу реализации алгоритма отсечения линий Лианга–Барски.

6.9. Выведите преобразования симметрии, с помощью которых расчет точки пересечения в трех областях, показанных на рис. 6.16, можно перевести в другие шесть областей плоскости xy.

6.10. Разработайте схему алгоритма Николла–Ли–Николла для любой входной пары конечных точек.

6.11. Сравните число арифметических операций в алгоритме НЛН с вычислительной сложностью алгоритмов Коэна–Сазерленда и Лианга–Барски для нескольких различных ориентаций линии относительно отсекающего окна.

6.12. Примените алгоритм Лианга–Барски к отсечению многоугольников.

6.13. Задайте подробную схему отсечения многоугольников по Уэйлеру–Азертону, предполагая, что отсекающее окно — прямоугольник стандартной ориентации.

6.14. Разработайте алгоритм обработки многоугольников по Уэйлеру–Азертону, где отсекающее окно может быть любым выпуклым многоугольником.

6.15. Разработайте алгоритм обработки многоугольников по Уэйлеру–Азертону, где отсекающее окно может быть любым заданным многоугольником (выпуклым или вогнутым).

6.16. Напишите процедуру отсечения эллипса стандартной ориентации прямоугольным окном.

6.17. Предполагая, что все символы строки текста имеют одинаковую ширину, разработайте алгоритм отсечения текста согласно стратегии отсечения символов по принципу “все или ничего”.

6.18. Придумайте алгоритм обработки текста, отсекающий отдельные символы, предполагая, что символы определены пиксельной сеткой заданного размера.