390-393 Роджерс

6.4. Метод Гуро

Метод закраски, который основан на интерполяции интенсивности и известен как метод Гуро (по имени его разработчика), позволяет устранить дискретность изменения интенсивности. Процесс закраски по методу Гуро осуществляется в четыре этапа:

1. Вычисляются нормали ко всем полигонам.
2. Определяются нормали в вершинах путем усреднения нормалей по всем полигональным граням, которым принадлежит вершина (рис 6.3).

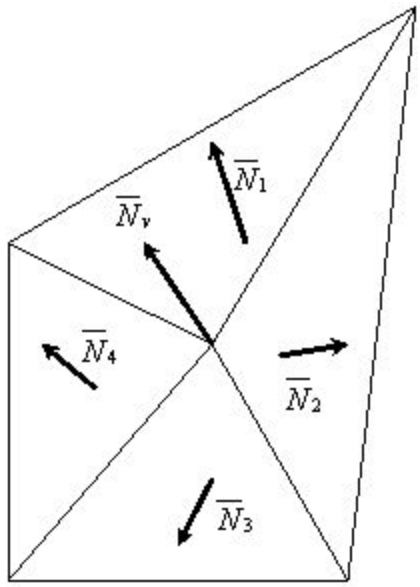


Рис. 6.3. Нормали к вершинам: $\overline{N}_v = (\overline{N}_1 + \overline{N}_2 + \overline{N}_3 + \overline{N}_4 + \overline{N}_v)/4$

3. Используя нормали в вершинах и применяя произвольный метод закраски, вычисляются значения интенсивности в вершинах.
4. Каждый многоугольник закрашивается путем линейной интерполяции значений интенсивностей в вершинах сначала вдоль каждого ребра, а затем и между ребрами вдоль каждой сканирующей строки (рис. 6.4).

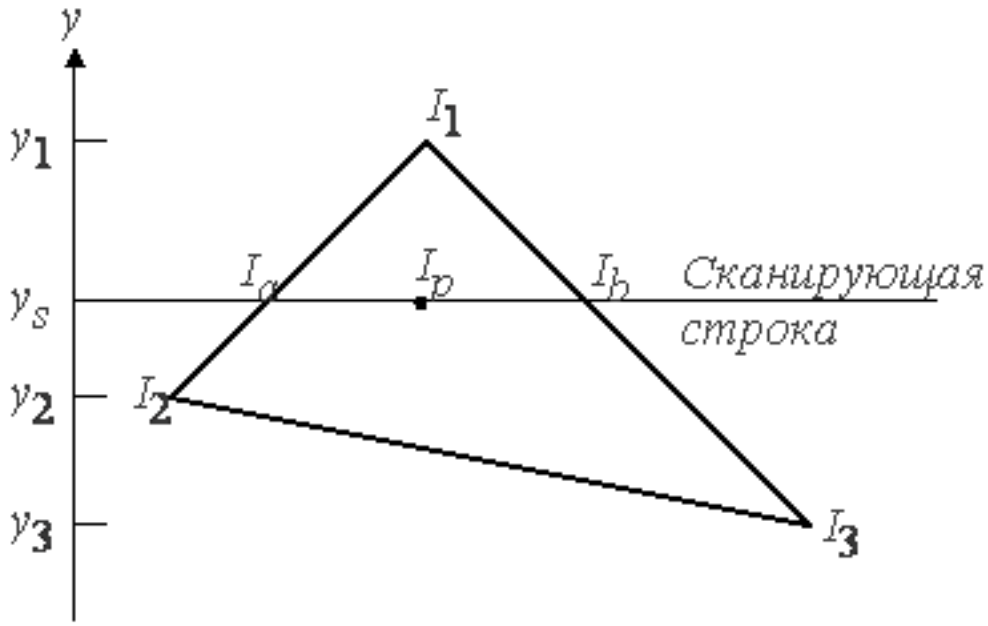


Рис. 6.4. Интерполяция интенсивностей

Интерполяция вдоль ребер легко объединяется с алгоритмом удаления скрытых поверхностей, построенным на принципе построчного сканирования. Для всех ребер запоминается начальная интенсивность, а также изменение интенсивности при каждом единичном шаге по координате *y*, Заполнение видимого интервала на сканирующей строке производится путем интерполяции между значениями интенсивности на двух ребрах, ограничивающих интервал (рис 6.4).

$$I_a = I_1 \quad \frac{y_s - y_2}{y_1 - y_2} + I_2 \quad \frac{y_1 - y_s}{y_1 - y_2};$$

$$I_b = I_1 \quad \frac{y_s - y_3}{y_1 - y_3} + I_3 \quad \frac{y_1 - y_s}{y_1 - y_3};$$

$$I_p = I_a \quad \frac{x_b - x_1}{x_b - x_a} + I_b \quad \frac{x_p - x_a}{x_b - x_a} \cdot$$

Для цветных объектов отдельно интерполируется каждая из компонент цвета.

Располагая средствами вычисления векторов нормали, можно при заданном расположении источников света и наблюдателя применить рассмотренные модели ко всем точкам поверхностей объектов сцены. Использование полигональной модели для тонирования поверхностей объектов сцены существенно уменьшает объем вычислений. Каждый многоугольник в такой сети - плоский, и вычислить компоненты вектора нормали к нему не представляет особого труда. Рассмотрим три метода закрашивания многоугольников: плоское, закрашивание по методу Гуро, и закрашивание по методу Фонга [21, 24].

Плоское закрашивание. Если поверхность плоская, то вектор нормали *n* остается постоянным для всех точек этой поверхности. Для реализации алгоритма закрашивания нужно вместо расположения источника задавать направление на источник. Если вектор нормали постоянен для всех точек многоугольника, то все необходимые вычисления для его закрашивания можно выполнить только один раз и применить результаты ко всем точкам этого многоугольника.

Этот метод получил название плоского закрашивания. На изображении, сформированном алгоритмом плоского закрашивания, четко видна разница в оттенках цвета отдельных многоугольников сети.

Рис.10.4. Плоское закрашивание

Закрашивание по методу Гуро. Метод Гуро основывается на идее закрашивания каждой плоской грани не одним цветом, а плавно изменяющимися оттенками, вычисляемыми путем интерполяции цветов примыкающих граней. Закрашивание граней по методу Гуро осуществляется в четыре этапа.

Шаг 1. Вычисляются нормали к каждой грани.

Шаг 2. Определяются нормали в вершинах. Нормаль в вершине определяется усреднением нормалей примыкающих граней (рис. 10.5).

.

Шаг 3. На основе нормалей в вершинах вычисляются значения интенсивности в вершинах согласно выбранной модели отражения света.

Шаг 4. Закрашиваются полигоны граней цветом, соответствующим линейной интерполяции значений интенсивности в вершинах.

Недостаток метода то, что если источник света проецирутся в плоскость многоугольника, то, используя этот метод заливки, будет получен результат рис.8 (1), хотя должно быть рис.8 (2).

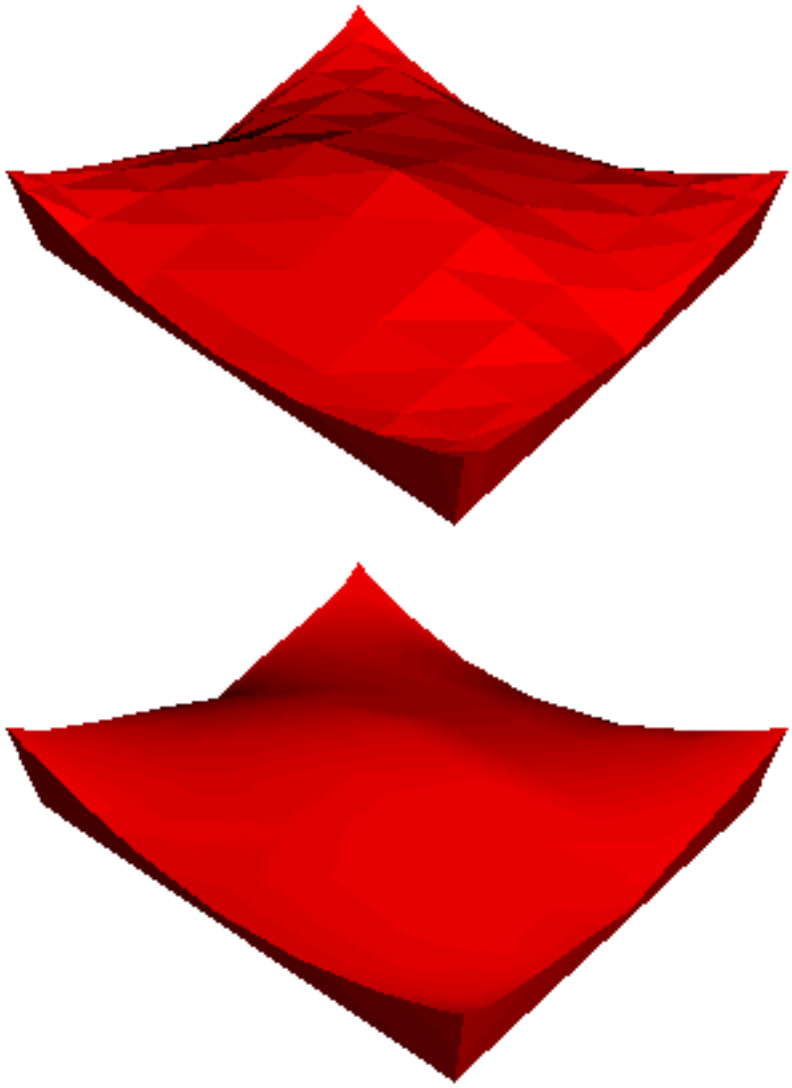
картинки вікі

Особенности моделирования света: Затенение по Гуро (Gourad Shading)

Затенение по Гуро — это метод линейной интерполяции освещенности в пределах одного полигона. Он был изобретен в 1971 и носит имя своего изобретателя. Это простой и эффективный метод придания ощущения изогнутости для ровного полигона. Этот метод также часто используется для сокращения глубины прорисовываемой сцены путем имитации исчезновения удаленных объектов в тумане.

! Полигоном в трехмерной графике принято считать участок поверхности, имеющий как минимум три вершины лежащие в одной плоскости. Собственно говоря, полигон

может иметь неограниченное количество вершин, но в трехмерной компьютерной графике в подавляющем большинстве случаев разработчики используют именно треугольные полигоны. Во-первых, три вершины всегда однозначно определяют положение плоскости в пространстве, а четвертая вершина может лежать вне пределов плоскости, и ее положение придется всегда проверять, что вызовет дополнительные трудности. Во-вторых, из треугольников всегда можно получить любой другой многоугольник (прим. переводчика).



Слева вы видите изогнутую поверхность (состоящую из множества треугольных полигонов, или попросту — треугольников), отвизуализированную на экран обычным образом, где каждая треугольная грань (полигон) имеет равномерную освещенность(flat shading), справа та же поверхность, но с применением затенения Гуро (Gourad shading). Несомненно, второй вариант выглядит куда более привлекательно, поэтому рассмотрим один из вариантов его реализации.*(Обращаю внимание уважаемых читателей, что вариантов реализации метода Гуро существует*

несколько, ниже приводится собственный вариант, разработанный и опробованный автором.)

В отличие от равномерного заполнения, в случае метода Гуро вы прежде всего обязаны определить и просчитать одним из известных способов приведенный вектор нормали для каждой вершины полигона. Получив нормаль, мы можем определить shade (степень затенения или освещенность) в вершинах, а уже затем нам остается только провести интерполяцию освещенности по полигону.

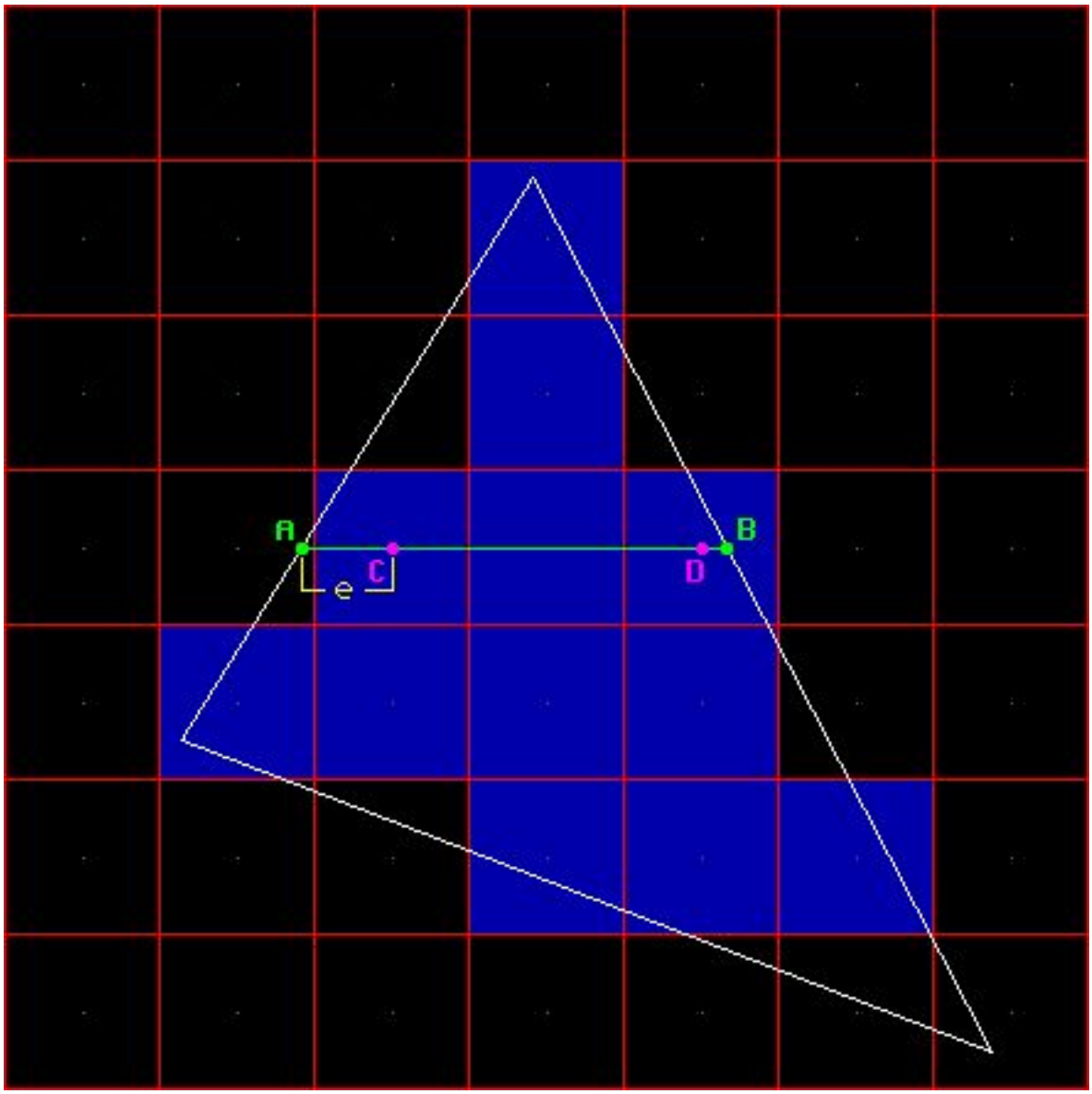
При реализации затенения Гуро, по методу, изложенному ниже, необходимо ограничится использованием только треугольных полигонов. Так как иногда полигоны будут выглядеть не так, как хотелось бы, если они имеют более трех вершин с различными оттенками в каждой из них. С треугольными полигонами все будет работать отлично.

Ниже мы рассмотрим подробнее фазу интерполяции освещения по полигону.

Визуализация полигона и интерполяция освещенности будет проведена путем scanline converting вдоль координаты X.

Будем предполагать, что освещенность в вершинах уже вычислена, и допустив, что вдоль кромок полигона она изменяется линейно, вычислить ее значение в точках A и B на каждое изменение координаты Y будет несложно. Конечно, можно и непосредственно определить нормаль в данных точках и тут же вычислить освещенность... — вариантов несколько, но суть в том, что перед началом дальнейших вычислений мы знаем освещенность в точках A и B.

Перед тем, как первая горизонтальная линия будет просчитана, сделаем некоторые приготовления. Рисунок ниже поможет нам разобраться с тем, что же происходит на самом деле.



Небольшой полигон всего в несколько квадратных пикселей изображен с равномерным заполнением только для простоты. Мы хотим просчитать линию A-B. Но мы имеем дело с пикселизованным экраном, поэтому мы сможем вывести на экран только линию, состоящую из трех пикселей C- D. Центр первого пикселя, C, не совпадает с центром точки A, а центр пикселя D не совпадает с точкой B.

Можно посчитать, что смещения в доли пикселя можно и не учитывать, но это будет справедливо только для больших полигонов, а вот для небольших это смещение учитывать надо. Особенно, если на полигон будет наложена текстура.

В нашем случае мы произведем эти приготовления:

Сначала определим градиент изменения освещенности (shade) по всей длине линии. Произведем это обычным способом.

Gradient = (Bs — As) / (Bx — Ax);

где:

- As и Bs — оттенок в точках A и B

- Bx и Ax — значения X в точках A и B соответственно

градиент показывает относительную величину изменения освещенности (shade) в пересчете на единицу изменения координаты X.

Теперь мы определим точное значение (shade) в точке C.

Cs = (1 — frac(Ax)) * Gradient;

величина **frac(Ax)** определяет смещение **e** , см. рисунок.

Теперь мы готовы к визуализации одной горизонтальной полосы из состава нашего полигона Гуро. Процесс включает в себя просчет оттенка каждого пикселя в линии и выводе его на устройство визуализации. Это достаточно простой процесс, так как величина изменения оттенка (градиент) линейна на всей протяженности выводимой линии. Вот пример псевдокода:

```
Shade = Cs
loop X from Cx to Dx
plot pixel at (X,Y) with colour Shade //вывод пикселя
Shade = Shade + Gradient
End of X loop
```

Как обычно, можно значительно ускорить скорость работы данного алгоритма, если реализовать его на ассемблере. Алгоритм, приведенный ниже, работает только в 8-битном режиме экрана и приведен исключительно с целью простоты демонстрации. Он далек от совершенства, но все же обладает очень приемлемой скоростью вывода. Для всех остальных режимов работы можно составить аналогичные алгоритмы.

Метод реализует возможность x86 процессоров трактовать 32-битный регистр как два 16-битных и, соответственно, 16-битный регистр как два 8-битных. Данный код использует старший байт регистра как целочисленную часть интерполируемого оттенка, а младший байт — для хранения дробной части. Такое решение позволяет выполнить интерполяцию, используя только целочисленные операции, освобождая сопроцессор для выполнения других задач.

Показанный внутренний цикл производит просчет только одной пиксельной линии полигона. Он использует 32-битные указатели на область памяти, отображающую экран.

Ввиду того, что значение величины освещенности определяется значением с фиксированной точкой (целая и дробные части хранятся в разных регистрах), мы достигаем повышенной точности вычислений, не вызывая дополнительных операций с плавающей точкой.

Для демонстрации фрагмента кода определим, что:

- length = Dx — Cx + 1 ;длина выводимой линии
- ScreenPtr = y*ScreenWidth + Cx ;указатель

А теперь сам внутренний цикл:

- CX = length
- AX = Cs * 256
- BX = Gradient

- EDI = ScreenPtr
-
- top:
    - mov [edi], ah ;write the pixel to the screen
    - inc edi ;next pixel
    - add ax, bx ;update shade
    - dec cx ;decrement counter
    - jnz top ;loop if more pixels

Влияние освещения при реализации затенения Гуро

Несмотря на то, что затенение Гуро далеко от идеала, оно все же позволяет создавать значительно более реалистичные поверхности. Недостатки затенения Гуро начинают проявляться тогда, когда вы пытаетесь совместить вычисления освещенности с затенением больших по размеру полигонов.

Представьте себе — вы имеете большой полигон, освещенный единственным источником света, расположенным в центре полигона. Освещенность, создаваемая источником света в вершинах полигона, будет очень малой ввиду значительной удаленности от источника света. Выведенный на экран полигон будет темным, хотя это не правильно, ведь центр полигона — освещен! Этот недостаток вы можете заметить, например, в игре Descent. Вспышки от выстрелов ярко освещают углы стен и практически не влияют на изменение освещения, если вы выстреливаете в середину стены или пола.

(И наоборот, большой полигон будет всегда полностью освещен если с его противоположных краев стоит по источнику света, даже несмотря на то, что источники маломощны и середина полигона просто обязана быть в тени)

Однако, если вы используете большое количество небольших полигонов совместно с отдаленным источником света, то затенение Гуро может выглядеть очень и очень неплохо. Практически — чем меньше размер полигона, тем точнее и качественней будет сглаживание, тем больше оно будет похоже по качеству на затенение по Фонгу.