实心三角形

在上一章中，我们迈出了绘制简单形状的第一步 - 即直线段 - 仅使用基于简单数学的算法。在本章中，我们将重用一些数学来绘制更有趣的东西：一个填充三角形。

`PutPixel`

绘制线框三角形

我们可以用这种方法画出三角形的轮廓：`DrawLine`

```
DrawWireframeTriangle (P0, P1, P2, color) {
    DrawLine(P0, P1, color);
    DrawLine(P1, P2, color);
    DrawLine(P2, P0, color);
}
```

这种轮廓称为*线框*，因为它看起来像一个由电线组成的三角形，如图 7-1 所示。

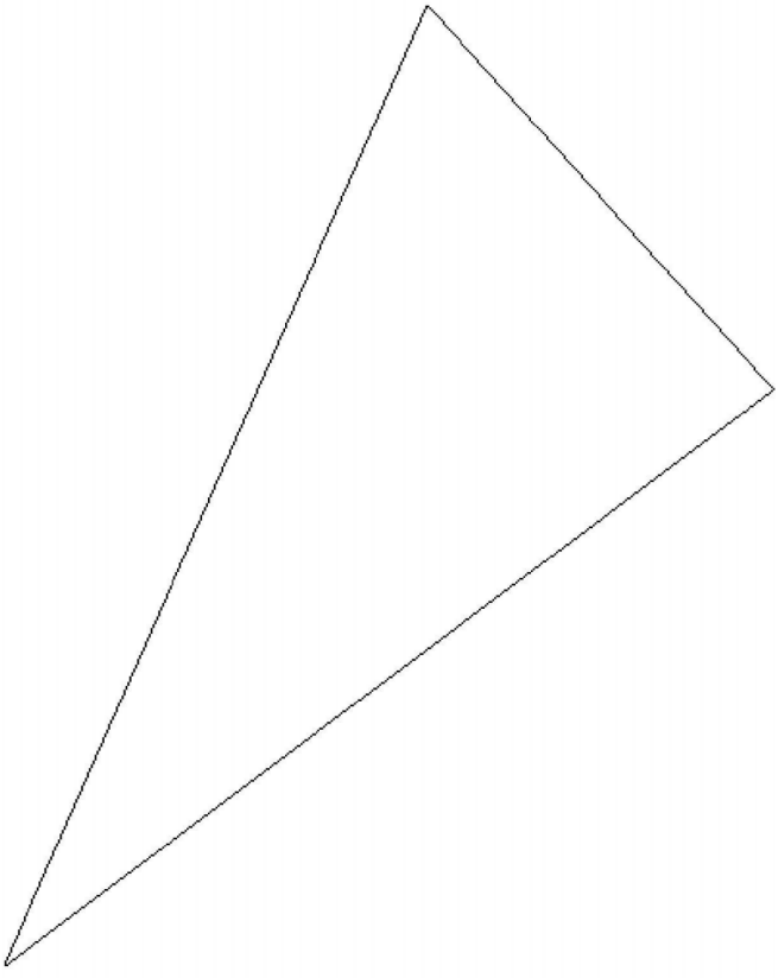

图 7-1：具有顶点 (–200、–250)、 (200，50) 和 (20，250) 的线框三角形

这是一个充满希望的开始！接下来，我们将探讨如何用颜色填充该三角形。

绘制填充三角形

我们想画一个用我们选择的颜色填充的三角形。与计算机图形学中经常出现的情况一样，解决此问题的方法不止一种。我们将绘制填充三角形，方法是将它们视为水平线段的集合，这些线段在绘制在一起时看起来像一个三角形。图 7-2 显示了如果我们可以看到各个段，这样的三角形会是什么样子。

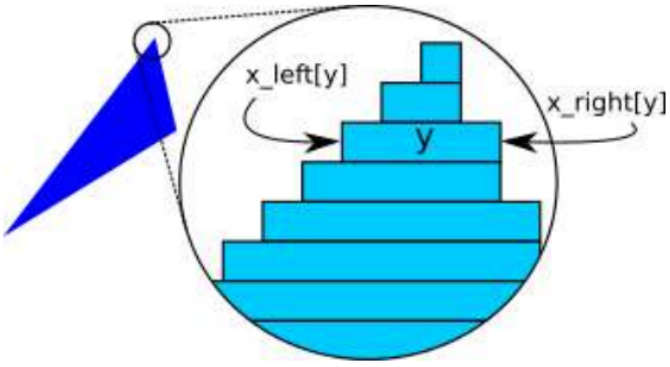

图 7-2：使用水平线绘制填充三角形

以下是我们想要做的非常粗略的第一近似值：

```
for each horizontal line y between the triangle's top and bottom
    compute x_left and x_right for this y
    DrawLine(x_left, y, x_right, y)
```

让我们从“三角形的顶部和底部之间”开始。三角形由其三个顶点定义P_0,P_1和P_2.如果我们通过增加值来对这些点进行排序y，这样$y_0 \leq y_1 \leq y_2$，则值的范围y三角形占据的是$[y_0, y_2]$：

```
if y1 < y0 { swap(P1, P0) }
if y2 < y0 { swap(P2, P0) }
if y2 < y1 { swap(P2, P1) }
```

以这种方式对顶点进行排序使事情变得更容易：完成此操作后，我们总是可以假设P_0是三角形的最低点，并且P_2是最高的，所以我们不必处理每一个可能的排序。

接下来，我们必须计算 和 数组。这有点棘手，因为三角形有三个边，而不是两条边。但是，仅考虑x_leftx_right$y$，我们总是有“高大”的一面P_0自P_2，以及来自P_0自P_1和P_1自P_2.

有一个特殊情况，$y_0 = y_1$或$y_1 = y_2$—也就是说，当三角形的一条边是水平的时。发生这种情况时，其他两侧具有相同的高度，因此任何一侧都可以被视为“高”一侧。我们应该选择右侧还是左侧？幸运的是，这并不重要;该算法将支持从左到右和从右到左的水平线，因此我们可以坚持我们的定义，即“高”边是来自P_0自P_2.

的值将来自高边或连接短边;的值将来自另一个集合。我们将从计算x_rightx_left$x$对于三个方面。由于我们将绘制水平线段，因此我们只想要一个值x对于y;这意味着我们可以通过使用 来计算这些值Interpolate; 作为自变量和x作为因变量

```
x01 = Interpolate(y0, x0, y1, x1)
x12 = Interpolate(y1, x1, y2, x2)
x02 = Interpolate(y0, x0, y2, x2)
```

这x其中一侧的值位于 ;另一端的值来自 和 的串联。请注意，和 中有一个重复的值x02x01x12x01x12x的值y_1既是 的最后一个值，也是x01

的第一个值。我们只需要去掉其中一个（我们任意选择 的最后一个值），然后连接数组：x12x01

```
remove_last(x01)
x012 = x01 + x12
```

我们终于有了 和 ，我们需要确定哪个是，哪个是。为此，我们可以选择任何水平线（例如，中间的一条）并比较其x02x012x_leftx_right$x$和 中的值：如果x02x012x值 in 小于 in 的值，那么我们知道一定是 ;否则，它必须是.x02x012x02x_leftx_right

```
m = floor(x02.length / 2)
if x02[m] < x012[m] {
    x_left = x02
    x_right = x012
} else {
    x_left = x012
    x_right = x02
}
```

现在我们有了绘制水平线段所需的所有数据。我们可以为此使用。但是，这是一个非常通用的函数，在这种情况下，我们总是绘制水平的、从左到右的线条，因此使用简单的循环更有效。这也让我们对绘制的每个像素有更多的“控制”，这将在后续章节中特别有用。DrawLineDrawLinefor

示例 7-1 已完成 。DrawFilledTriangle

```
DrawFilledTriangle (P0, P1, P2, color) {
  ①// Sort the points so that y0 <= y1 <= y2
   if y1 < y0 { swap(P1, P0) }
   if y2 < y0 { swap(P2, P0) }
   if y2 < y1 { swap(P2, P1) }

  ❷// Compute the x coordinates of the triangle edges
   x01 = Interpolate(y0, x0, y1, x1)
   x12 = Interpolate(y1, x1, y2, x2)
   x02 = Interpolate(y0, x0, y2, x2)

  ❸// Concatenate the short sides
   remove_last(x01)
   x012 = x01 + x12
```

```
    ④// Determine which is left and which is right
    m = floor(x012.length / 2)
    if x02[m] < x012[m] {
        x_left = x02
        x_right = x012
    } else {
        x_left = x012
        x_right = x02
    }

    ⑤// Draw the horizontal segments
    for y = y0 to y2 {
        for x = x_left[y - y0] to x_right[y - y0] {
            canvas.PutPixel(x, y, color)
        }
    }
}
```

示例 7-1：绘制填充三角形的函数

让我们看看这里发生了什么。该函数以任意顺序接收三角形的三个顶点作为参数。我们的算法需要它们按从下到上的顺序排列，所以我们以这种方式对它们进行排序❶。接下来，我们计算三边 ❷ 的每个值的值，并从两个“短”边 ❸ 连接数组。然后我们弄清楚哪个是，哪个是❹。最后，对于三角形顶部和底部之间的每个水平段，我们得到它的左右坐标，并逐像素绘制该段❺。`xyx_leftx_rightx`

图 7-3 显示了结果;出于验证目的，我们调用，然后使用相同的坐标但不同的颜色。尽可能验证结果 - 这是查找代码中错误的非常有效的方法!

`DrawFilledTriangleDrawWireframeTriangle`

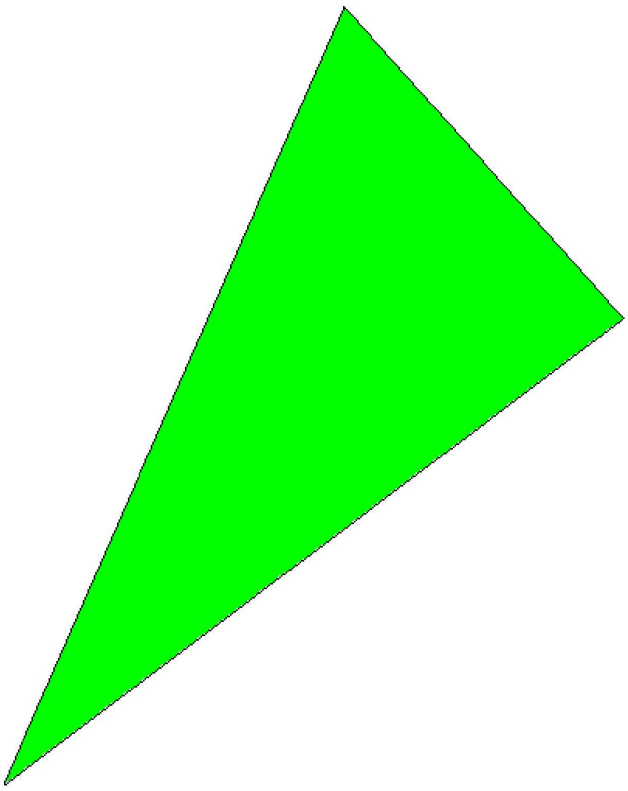

图 7-3：填充三角形，带有用于验证的线框边缘

您可能会注意到三角形的黑色轮廓与绿色内部区域不完全匹配;这在三角形的右下边缘尤其明显。这是因为是计算DrawLine$y = f(x)$对于该边缘，但正在计算DrawTriangle $x = f(y)$，并且由于四舍五入，这可能会产生略有不同的结果。这是我们愿意接受的近似误差，以使我们的渲染算法快速。

总结

在本章中，我们开发了一种算法，用于在画布上绘制填充三角形。这是绘制线段的升级。我们还学会了将三角形视为一组可以单独处理的水平段。

在下一章中，我们将扩展数学和算法，以绘制一个填充有颜色渐变的三角形;算法背后的数学和推理将是本书中开发的其他功能的关键。