6 Wie kann man (perspektivische) Objekte gemustert darstellen?

In diesem Kapitel zeigen wir Ihnen, wie man Objekte mit Füllmustern versehen kann. Dies allein ist in GDI+ nicht weiter schwer, interessant wird das Ganze aber bei perspektivisch dargestellten Objekten. Nehmen wir exemplarisch eine Ziegelsteinmauer: Sie soll perspektivisch nach hinten verlaufen und – wenn wir schon dabei sind – auch noch perspektivisch beleuchtet sein (siehe *Abbildung 6.1*).

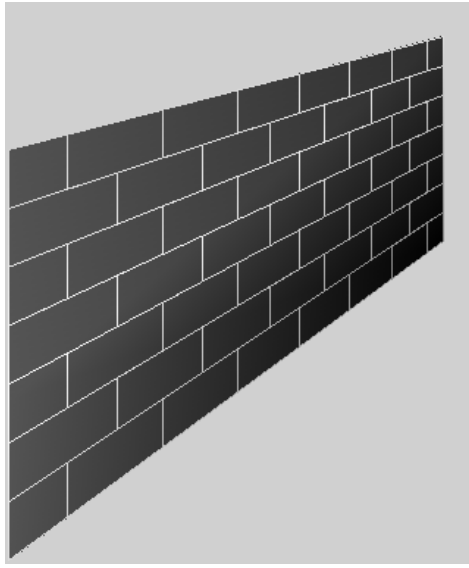

Abbildung 6.1: Perspektivische Ziegelsteinmauer (Beleuchtung besser sichtbar im Farbteil, Abbildung F.6)

Bei der Darstellung unserer Ziegelsteinmauer treffen wir gleich auf mehrere Fragen:

- Wie stellt man eine Ziegelsteinmauer dar?
- Wie stellt man die Mauer perspektivisch dar?
- Wie erhält man eine geeignete Beleuchtung?

Diese Fragen wollen wir nun der Reihe nach beantworten. Dabei erhalten Sie gleichzeitig eine generelle Übersicht über die Möglichkeiten von GDI+, Objekte mit Füllmustern zu versehen.

Zunächst benötigen wir aber eine Lösungsidee für die erste Frage:

Eine Mauer kann man sich als rotes Rechteck vorstellen, das mit einem weißen Mauerfugen-Muster ausgefüllt ist. Man kann das Fugen-Muster auf zwei Arten erzeugen:

- Zeichnen der weißen Linien über die ganze Mauerfläche
- Zeichnen des Musters für *einen* Ziegelstein und Wiederholung des Musters über die ganze Fläche (Damit die typischen, horizontal versetzten Ziegelsteinreihen entstehen können, muss das Wiederholmuster genauer gesagt 1 ½ Ziegelsteine darstellen).

In *Abbildung 6.2* sind beide Möglichkeiten skizziert (im rechten Bild ist das Wiederholmuster gestrichelt dargestellt):

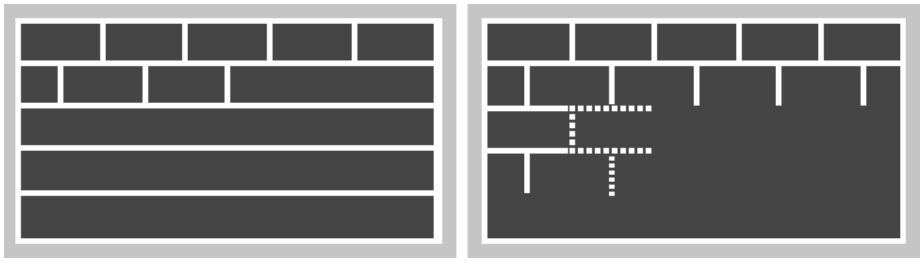

Abbildung 6.2: Erzeugen der Ziegelsteinfugen (siehe auch Farbteil, Abbildung F.5)

In GDI+ werden Füllmuster über ***Brush***-Objekte realisiert. Es gibt für beide der oben dargestellten Lösungsideen Möglichkeiten zur Umsetzung, die wir Ihnen im Folgenden vorstellen. Wir beginnen mit den Wiederholmustern.

6.1 Hatch Brush

Der *Hatch Brush* (abgeleitet von *to hatch*: schraffieren) füllt eine Fläche mit einem vorgegebenen Muster (*HatchStyle*), das horizontal und vertikal über die ganze Fläche wiederholt wird. Die Muster sind in GDI+ vordefinierte 8 x 8 Pixel große Bitmaps und eines davon ist ein Ziegelsteinmuster (*HatchStyle.HorizontalBrick*).

In der von uns benutzten Variante der Deklaration des *Hatch Brushes* können wir außerdem die Vorder- und Hintergrundfarbe des Musters festlegen:

```
Public Sub New(ByVal hs As HatchStyle, ByVal foreColor As Color, ByVal backColor As Color)
```

Anhand des vordefinierten Ziegelsteinmusters wollen wir einen ersten Versuch für unsere Mauer starten und lernen dabei gleich eine Eigenart des *Hatch Brushes* kennen. Schauen wir uns den Code für eine – zunächst noch nicht perspektivische – Mauer an:

```
Imports System.Drawing.Drawing2D         'Namespace des Hatch Brush

Public Class frmWall
   Inherits Form

   Private Sub frmWall_Paint(ByVal sender As Object, _
                             ByVal e As PaintEventArgs) Handles MyBase.Paint

      Dim recWall As New Rectangle(100, 100, 150, 300)
      Dim hbWall As New HatchBrush(HatchStyle.HorizontalBrick, Color.White, Color.Firebrick)

      e.Graphics.FillRectangle(hbWall, recWall)

      hbWall.Dispose()

   End Sub

End Class
```

Im *Paint*-Ereignis des Formulars, auf das wir die Mauer zeichnen, definieren wir uns (mit beliebigen Koordinaten) ein Rechteck. Danach erzeugen wir einen *Hatch Brush* mit dem Ziegelsteinmuster und den Farben Weiß und Ziegelrot als Vorder- und Hintergrundfarbe. Über die *Graphics*-Methode *FillRectangle* füllen wir das Rechteck schließlich mit dem Brush.

Das Ergebnis sieht schon nicht schlecht aus (siehe *Abbildung 6.3*), auch wenn unsere Mauer durch die nur 8 x 8 Pixel große *Hatch Bitmap* sehr kleine Ziegelsteine enthält:

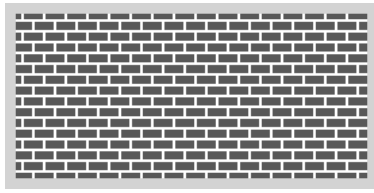

Abbildung 6.3: Hatch Brush Mauer

Schöner wäre es noch, wenn die Mauer nicht mit unvollständigen Ziegelsteinreihen beginnen und abschließen würde (in der Höhe und Breite).

Die Größe des *Hatch Brushes* lässt sich nicht ändern, wir können aber über die Eigenschaft ***RenderingOrigin*** des *Graphics*-Objekts die linke obere Ecke des *Hatch Brushes* an einem beliebigen Punkt im Koordinatensystem ausrichten. Der Default für den *Rendering Origin* ist der Ursprung des GDI+ Koordinatensystems auf dem Formular (genauer: der Client Area) und nicht etwa der Beginn der gezeichneten Fläche! Das Muster beginnt also in der linken oberen Ecke des Formulars und ist für das komplette Formular gültig. Die über *FillRectangle* angegebene Fläche öffnet sozusagen eine Durchsicht auf das Formularmuster.

Wir richten den *Hatch Brush* nun an der linken oberen Ecke unserer Mauer aus und passen den Code für das *Paint*-Ereignis entsprechend an:

```
Private Sub frmWall_Paint(ByVal sender As Object, ByVal e As PaintEventArgs)_
                        Handles MyBase.Paint

   Dim recWall As New Rectangle(100, 100, 153, 305)
   Dim hbWall As New HatchBrush(HatchStyle.HorizontalBrick, Color.White, Color.Firebrick)

   e.Graphics.RenderingOrigin = New Point(100, 100)
   e.Graphics.FillRectangle(hbWall, recWall)

   hbWall.Dispose()
End Sub
```

Die Ziegelsteine kann man auf diesem Weg natürlich nur an zwei Seiten der Mauerfläche ausrichten. Die beiden anderen Seiten müssen über die Rechteckgröße der Mauer angepasst werden. *Abbildung 6.4* zeigt das Ergebnis:

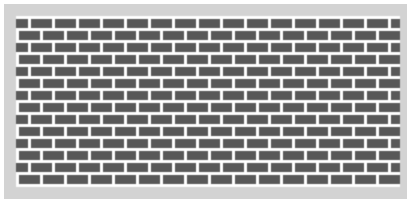

Abbildung 6.4: Hatch Brush Mauer mit ausgerichtetem Muster (siehe auch Farbteil, Abbildung F.5)

Die Ausrichtung des Musters ist wie gezeigt unabhängig von der Form der Fläche, die es ausfüllen soll, und lässt sich auch von Transformationen auf den Flächen nicht beeindrucken. Das Ergebnis einer Anwendung des *Hatch Brushes* auf eine perspektivisch veränderte Mauerfläche[1] (siehe *Abbildung 6.5*) lässt sich damit schon erahnen.

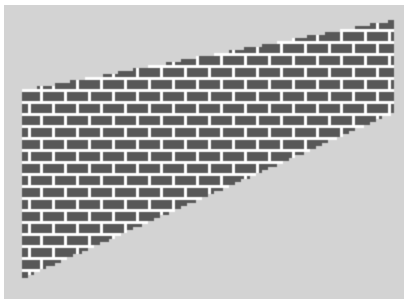

Abbildung 6.5: Perspektivische Hatch Brush Mauer (siehe auch Farbteil, Abbildung F.5)

[1] Nähere Informationen zum perspektivischen Zeichnen finden Sie in *Kapitel 5*.

Fazit:

Der Ziegelstein *Hatch Brush* bietet eine einfache Möglichkeit zur Darstellung einer Mauer. Er kann aber nur dann eingesetzt werden, wenn die Mauer horizontal stehen soll.

Neben den *Hatch Brushes* bietet GDI+ für Wiederholmuster auch die Verwendung von *Texture Brushes* an. Versuchen wir es im nächsten Schritt damit.

6.2 Texture Brush

Texture Brushes füllen Flächen ebenfalls mit einem Muster, das innerhalb der Fläche horizontal und vertikal wiederholt wird. Es gibt aber zwei Hauptunterschiede zu den *Hatch Brushes*:

1. Als Muster (Textur) kann ein beliebiges Image-Objekt dienen.
2. Auf Texture Brushes können Transformationen angewendet werden.

Leider währt auch hier die Freude nur kurz. Als Textur können wir zwar das *Hatch Brush* Muster für Ziegelsteine leicht nachbilden, die verfügbaren Transformationen Verschieben (*TranslateTransform*), Skalieren (*ScaleTransform*) und Rotieren (*RotateTransform*) bieten aber keine Möglichkeit einer nicht linearen Transformation, die für eine perspektivische Darstellung notwendig wäre (vgl. Abschnitt *Darstellung der Mauerfugen*).

Fazit:

Texture Brushes eignen sich, wenn:

- kein passender Hatch Brush vorhanden ist.
- ein Hintergrundmuster benötigt wird, das mehr als zwei Farben enthält (*Hatch Brushes* bieten lediglich eine Vorder- und Hintergrundfarbe).
- das Muster in verschiedenen Transformationen benötigt wird – außer eben bei einer nicht linearen Transformation, die auch für Texture Brushes nicht verfügbar ist.

Der Versuch, unsere perspektivische Mauer über GDI+ einfach aus den einzelnen Mauersteinen automatisch zusammensetzen zu lassen, hat also leider nicht funktioniert. Es bleibt uns nichts anderes übrig, als die Ziegelsteinfugen selbst zu konstruieren. Dazu verwenden wir *Path Gradient Brushes*, die wir gleich näher kennen lernen. Zusätzlich setzen wir die Grundlagen über Fluchtpunkte und perspektivisches Zeichnen aus *Kapitel 5* voraus.

6.3 Path Gradient Brush

Die hier verwendete Funktionsweise des *Path Gradient Brushes* lässt sich mit Hilfe von *Abbildung 6.6* demonstrieren.

Bei der Deklaration des *Path Gradient Brushes* übergibt man in der Regel lediglich ein *Polygon* (als Punktarray mit beliebig vielen Punkten) oder einen *Graphics Path*:

```
Public Sub New(ByVal points() As Point)
Public Sub New(ByVal path As GraphicsPath)
```

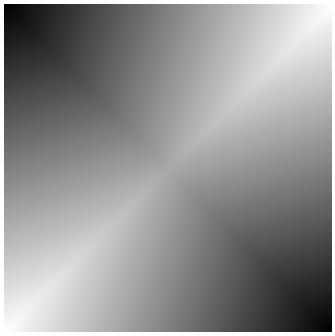

Abbildung 6.6: Beispiel eines Path Gradient Brush (besser sichtbar im Farbteil, Abbildung F.7)

Daneben wird das Aussehen des *Path Gradient Brushes* im Wesentlichen durch seine Eigenschaften ***CenterPoint***, ***CenterColor*** und ***SurroundColors*** bestimmt. Die *SurroundColors* werden in einem Array angegeben, das (durch die Reihenfolge bestimmt) jedem Punkt im Polygon (oder Path) eine Farbe zuordnet.[2] GDI+ erzeugt dann zwischen dem *CenterPoint* und den Polygonpunkten einen Farbverlauf, der durch die *CenterColor* und der dem jeweiligen Polygonpunkt zugeordneten *SurroundColor* definiert wird. Der *CenterPoint* ist dabei nicht auf die Polygonfläche beschränkt, sondern kann auch außerhalb liegen.

In *Abbildung 6.6* haben wir das Polygon als Punktarray mit den Punkten [(10;10), (10;110), (110,110), (110,10)], das *Colorarray* mit den Farben [Red; Blue; Green; Yellow] definiert, der *CenterPoint* liegt bei (55;55) und die *CenterColor* ist White.

Mit dem Path Gradient Brush haben wir nun die Grundlage für die Beleuchtung der Ziegelsteinmauer. Wagen wir also einen neuen Versuch und wir versprechen Ihnen, dieser wird funktionieren! Wir gehen in zwei Schritten vor:

1. Darstellen der Mauerfläche (inklusive korrekter Beleuchtung)
2. Darstellen der Mauerfugen (auch hier muss die Beleuchtung berücksichtigt werden)

Darstellung der Mauerfläche

Kümmern wir uns zunächst um die Mauerfläche. Wir definieren ein Polygon, das die Eckpunkte der perspektivischen Mauerfläche enthält. Den Fluchtpunkt legen wir in den Punkt[3] (***2/3 CSW ; 0***), die vordere Mauerkante durch die Punkte (***0 ; CSH***) und (***0 ; CSH – 7 h_s***), wobei h_s die Höhe eines Mauersteins ist und die Mauer aus insgesamt sieben Ziegelsteinreihen bestehen soll. Daraus ergeben sich für die beiden nach hinten verlaufenden Mauerkanten die folgenden Geradengleichungen (vgl. *Kapitel 5*):

$$Y_1 = -\frac{3\,CSH}{2\,CSW}X_1 + CSH \qquad Y_2 = -\frac{3\,(CSH - 7h_s)}{2\,CSW}X_2 + CSH - 7h_s$$

[2] Es sind auch weniger Farben als Punkte erlaubt. GDI+ verwendet dann die zuletzt angegebene Farbe für die Farbverläufe zu allen weiteren Polygonpunkten.

[3] CSW ist die Abkürzung für die Breite des Anwendungsformulars (**C**lient**S**ize.**W**idth), CSH für die Höhe (**C**lient**S**ize.**H**eight).

Die X-Koordinate für die hintere Mauerkante legen wir nun willkürlich auf den Wert ***CSW / 3*** fest. Die hinteren Mauerecken ergeben sich damit durch das Einsetzen dieses Wertes in die beiden Geradengleichungen zu (***CSW / 3 ; CSH / 2***) und (***CSW / 3 ; (CSH – 7 h_s) / 2***). Die Berechnung der Mauerfläche ist in *Abbildung 6.7* noch einmal veranschaulicht.

Das Polygon füllen wir mit einem *Path Gradient Brush*, dessen Farbverlauf von ziegelsteinrot im hell beleuchteten Bereich bis schwarz in der unbeleuchteten (hinteren) Ecke der Mauer reicht. Die Lichtquelle (CenterPoint des Path Gradient Brush) setzen wir – ebenfalls willkürlich – auf den Punkt ***(0 ; 3/4 * CSH)***.

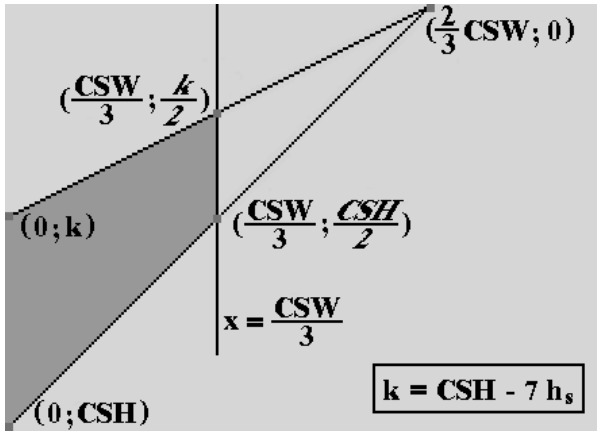

Abbildung 6.7: Berechnung der perspektivischen Mauerfläche

Schauen wir uns den Code für die Mauerfläche an:

```
Imports System.Drawing.Drawing2D    'Namespace d. PathGradientBrush

Public Class Form1
   Inherits Form

   Private Const DEF_WIDTH = 36
   Private Const DEF_HEIGHT = 25    'hs in obiger Berechnung

   Private CSW, CSH As Integer      'Abkürzungen

   Private Sub Wall_Paint(ByVal sender As Object, ByVal e As PaintEventArgs) _
                          Handles MyBase.Paint

      Dim pgWall As PathGradientBrush
      Dim ptaWallArea(3) As Point
      Dim ptCenter As New PointF(0, 3 * CSH \ 4)
      Dim colWall(3) As Color

      grfx = e.Graphics
      CSH = ClientSize.Height   :   CSW = ClientSize.Width
```

```
        '* Fläche für das Mauerwerk definieren *
        ptaWallArea(0).X = 0                          'Links unten
        ptaWallArea(0).Y = CSH
        ptaWallArea(1).X = 0                          'Links oben
        ptaWallArea(1).Y = CSH - (7 * DEF_HEIGHT)

        ptaWallArea(2).X = CSW \ 3                    'Rechts oben
        ptaWallArea(2).Y = ptaWallArea(1).Y \ 2

        ptaWallArea(3).X = CSW \ 3                    'Rechts unten
        ptaWallArea(3).Y = ptaWallArea(0).Y \ 2

        '* Farbverlauf für Mauerwerk definieren *
        colWall(0) = Color.Firebrick                  'Links unten
        colWall(1) = Color.Firebrick                  'Links oben
        colWall(2) = Color.DarkRed                    'Rechts oben
        colWall(3) = Color.Black                      'Rechts unten

        '* Path Gradient Brush erzeugen und konfigurieren *
        pgWall = New PathGradientBrush(ptaWallArea)
        pgWall.CenterPoint = ptCenter
        pgWall.CenterColor = Color.Red
        pgWall.SurroundColors = colWall

        '* Farbverlauf des Mauerwerks zeichnen *
        grfx.FillPolygon(pgWall, ptaWallArea)
        pgWall.Dispose()

        '[... hier folgt der Code für die Mauerfugen ...]

    End Sub

End Class
```

Wir definieren gemäß den angegebenen Geradengleichungen das Polygon, das die Mauer umrandet, sowie das Farbarray für den Path Gradient Brush. Nachdem alle Werte definiert sind, wird der Path Gradient Brush erzeugt. Die Anzeige des Ergebnisses ist dann dank GDI+ trivial.

Darstellung der Mauerfugen

Nun widmen wir uns dem schwierigeren Teil – der Konstruktion von Mauerfugen. Wie in *Kapitel 5* dargestellt, »verziehen« sich rechteckige Flächen bei der perspektivischen Darstellung. Durch die Ausrichtung an einem Fluchtpunkt laufen vorher parallele Linien aufeinander zu. Für die Konstruktion der Mauerfugen müssten wir also die perspektivische Breite und Höhe der einzelnen Ziegelsteine berechnen.[4]

[4] In *Kapitel 5* haben wir die Flächen nicht mit Mustern versehen. Deshalb sind wir dort auf dieses Problem nicht gestoßen.

Mathematisch kann man die perspektivische Fläche aber auch durch die (nicht lineare) Abbildung einer Rechteckfläche auf ein Polygon erzeugen. Wir können also eine »normale« Mauer konstruieren und diese durch eine entsprechende Abbildung auf die perspektivische Fläche übertragen (siehe *Abbildung 6.8*). Wenn Sie jetzt das (mathematische) Handtuch werfen wollen, können wir Sie beruhigen: GDI+ stellt uns genau diese Abbildung als so genannte ***Warp-Transformation***[5] zur Verfügung.

Die *Warp*-Transformation ist die einzige in GDI+ angebotene nicht lineare Transformation. Sie ist ausschließlich für den *Graphics Path* definiert und bildet eine Rechteckfläche auf eine Polygonfläche (das Polygon muss ebenfalls aus 4 Punkten bestehen) ab. Da die Figuren im *Graphics Path* nicht unbedingt rechteckig sind, verwendet man für die *Warp*-Abbildung normalerweise (aber nicht zwingend!) als Ausgangsrechteck das kleinste Rechteck, das alle Punkte des Path enthält. Dieses liefert er uns über die Methode ***GetBounds***. Als Zielpolygon können wir in unserem Fall das bereits für die Mauerfläche verwendete Polygon nutzen.

Was im ersten Moment recht kompliziert klingt, erklärt sich durch *Abbildung 6.8*:

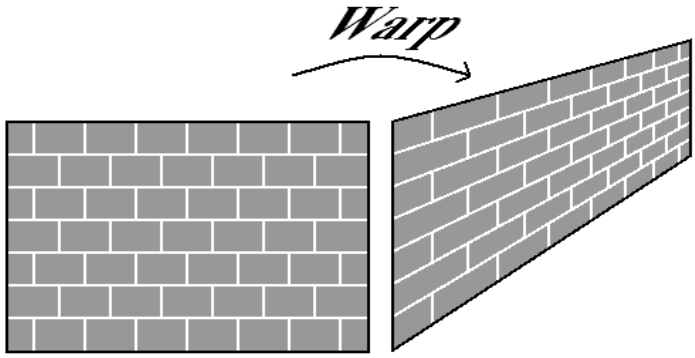

Abbildung 6.8: Perspektivische Mauer mit Warp-Transformation

Wir müssen die Mauerfugen also lediglich für eine horizontale Mauer erzeugen und dann die entsprechende *Warp*-Transformation auf die Mauerfläche durchführen. Die Größe der Ziegelsteine im linken Rechteck spielt dabei keine Rolle, lediglich die Proportionen müssen stimmen!

Der Code für die Erzeugung der Mauerfugen sieht damit wie folgt aus:

```
Private Sub Wall_Paint(ByVal sender As Object, ByVal e As PaintEventArgs) _
                       Handles MyBase.Paint

   '[... hier haben wir das Mauerwerk gezeichnet ...]

   Dim paBrick As New GraphicsPath()
```

[5] Engl. *warp*: sich verziehen; auch die Enterprise verzieht sich mit dem Warp-Antrieb in die unendlichen Weiten ...

```
    Dim ptaBrickArea(3) As PointF

    '* Mauerfugen in den GDI-Path <paBrick> einfügen *
    For i = 0 To 6
       AddWallRow(paBrick, (i Mod 2 = 0), CSH - 1 - (i * DEF_HEIGHT))
    Next

    '* ZielPolygon definieren *
    ptaBrickArea(0).X = ptaWallArea(1).X              'Links oben
    ptaBrickArea(0).Y = ptaWallArea(1).Y

    ptaBrickArea(1).X = ptaWallArea(2).X - 1          'Rechts oben
    ptaBrickArea(1).Y = ptaWallArea(2).Y + 1

    ptaBrickArea(2).X = ptaWallArea(0).X              'Links unten
    ptaBrickArea(2).Y = ptaWallArea(0).Y - 1

    ptaBrickArea(3).X = ptaWallArea(3).X - 1          'Rechts unten
    ptaBrickArea(3).Y = ptaWallArea(3).Y

    '* Mauerfugen in Zielkoordinaten transformieren *
    paBrick.Warp(ptaBrickArea, paBrick.GetBounds())

    [... hier folgt der Code für die richtige Beleuchtung ...]

End Sub

Private Sub AddWallRow(ByRef paPath As GraphicsPath, _
                       ByVal bTag As Boolean, ByVal y As Integer)

    Dim ptPolygon(3) As Point
    Dim nCurrX, nCount, i As Integer

    'Falls Reihe mit halbem Stein beginnt, muss ein Stein
    '  mehr verwendet werden
    nCount = IIf(bTag = True, 7, 6)  :  nCurrX = 0

    For i = 0 To nCount
       '* Koordinaten für Mauerstein berechnen *
       ptPolygon(0).X = nCurrX : ptPolygon(0).Y = y
       ptPolygon(1).X = nCurrX : ptPolygon(1).Y = y - DEF_HEIGHT

       'Falls <bTag>,an Anfang/ Ende nur halben Mauerstein einsetzen
       nCurrX = IIf((bTag And (i = 0)) Or (i = 7), _
                    nCurrX + (DEF_WIDTH \ 2), nCurrX + DEF_WIDTH)

       ptPolygon(2).X = nCurrX : ptPolygon(2).Y = ptPolygon(1).Y
       ptPolygon(3).X = nCurrX : ptPolygon(3).Y = ptPolygon(0).Y
```

```
        '* Mauerstein in GDI-Path einfügen *
        paPath.AddPolygon (ptPolygon)
    Next

End Sub
```

Wenn Sie den Code analysiert haben, dürften Ihnen zwei Dinge aufgefallen sein:

- Wir fügen in der Prozedur *AddWallRow* keine Linien in den Path ein, sondern Polygone, die jeweils einen Ziegelstein repräsentieren. Sie haben völlig Recht, damit zeichnen wir jede Menge Linien doppelt. Wir haben diesen Ansatz aus Gründen der Wiederverwertbarkeit für unsere Beispiele gewählt, Sie können aber ohne jede Einschränkung die Mauerfugen auch über Linien zeichnen.
- Die größere Verwunderung dürfte aber die Definition des Zielpolygons für die Warp-Transformation ausgelöst haben: Wieso wurde eine andere Reihenfolge bei den Polygonpunkten verwendet und wieso weichen die Koordinaten leicht von den Koordinaten des Mauerflächenpolygons ab?

  Bei der Verwendung des Polygons zum Zeichnen der Mauerfläche (Methode *FillPolygon*) müssen die Punkte der Reihe nach so angegeben werden, dass beim Ziehen einer Linie durch die Punkte das gewünschte Polygon entsteht (wie beim »Malen nach Zahlen«).

  Bei der Warp-Transformation ist leider eine andere Reihenfolge vorgeschrieben (siehe *Abbildung 6.9*). Eine Vertauschung der Punkte im Zielpolygon bewirkt eine Änderung der Richtung und Stärke der Verzerrung. Beispielsweise lässt sich damit auch eine Spiegelung einer Rechteckfläche realisieren (vgl. *Abbildung 6.9*).

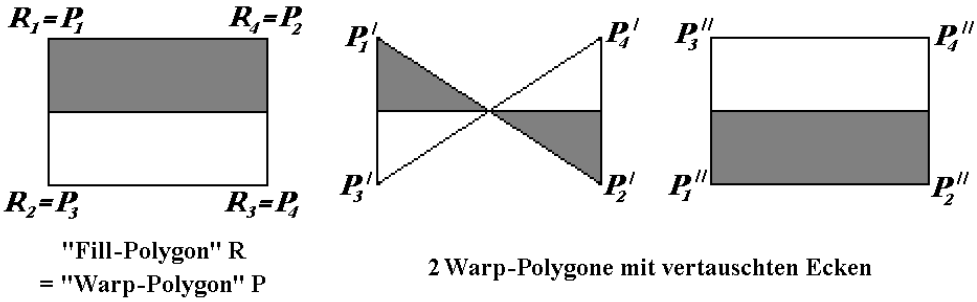

Abbildung 6.9: Reihenfolge der Ecken beim Warp

Die Abweichung bei den Koordinaten ist ein Phänomen, das wohl durch Rundungsungenauigkeiten bei der Berechnung der transformierten Punkte hervorgerufen wird. Experimentiert man ein wenig mit den Koordinaten, lassen die Veränderungen bei der anschließenden Transformation auf diese Ursache schließen. Die korrekten Werte haben wir also durch Experimentieren herausgefunden.

Beleuchtung der Mauerfugen

Ein Problem bleibt noch zu lösen: Um die Beleuchtung der Mauer realistisch zu simulieren, muss natürlich auch der Helligkeitsverlauf der Fugen entsprechend angepasst werden. Auch hier bietet uns GDI+ eine einfache Möglichkeit, diese Aufgabe zu lösen:

Die Fugenlinien haben wir in einen *Graphics Path* eingefügt (s.o.) und wir können sie damit über die *DrawPath*-Methode des *Graphics*-Objekts zeichnen. *Draw*-Methoden benötigen einen *Pen*, der als wesentliche Merkmale eine Zeichenfarbe und die Linienbreite enthält. GDI+ erlaubt es uns, als Zeichenfarbe auch einen Brush anzugeben (vgl. *Kapitel 2*). Wir verwenden nun einfach denselben *Path Gradient Brush*, wie wir ihn schon zum Zeichnen der Mauerfläche benutzt haben, und tauschen lediglich die Farben aus:

```
Private Sub Wall_Paint(ByVal sender As Object, ByVal e As PaintEventArgs) _
                      Handles MyBase.Paint

   '[... hier haben wir das Mauerwerk gezeichnet ...]
   '[... und die Mauerfugen berechnet ...]

   Dim pgBrick As PathGradientBrush
   Dim penBrick As Pen
   Dim colBrick(3) As Color

   '* Farbverlauf für Mauerfugen definieren *
   colBrick(0) = Color.Gainsboro : colBrick(1) = Color.LightGray
   colBrick(2) = Color.Gainsboro : colBrick(3) = Color.Gray

   '* Path Gradient Brush erzeugen und konfigurieren *
   pgBrick = New PathGradientBrush(ptaWallArea)
   pgBrick.CenterPoint = ptCenter
   pgBrick.CenterColor = Color.White
   pgBrick.SurroundColors = colBrick

   '* Mauerfugen zeichnen *
   penBrick = New Pen(pgBrick, 1)
   grfx.DrawPath(penBrick, paBrick)

   '* Aufräumen *
   penBrick.Dispose()  :  pgBrick.Dispose()  :  paBrick.Dispose()

End Sub
```

Das Polygon für den *Path Gradient Brush* ist nun wieder das gleiche Polygon wie bei der Mauerfläche, da der Brush nicht transformiert wird und damit keine Koordinatenkorrektur notwendig ist. Damit haben wir eine Lösung für eine perspektivisch gezeichnete Ziegelsteinmauer. In *Abbildung 6.10* sehen Sie noch einmal das Endergebnis:

Für eine perfekte Mauer fehlt natürlich noch die Mauertiefe, aber die sollte Ihnen nun keine besonderen Schwierigkeiten mehr bereiten.

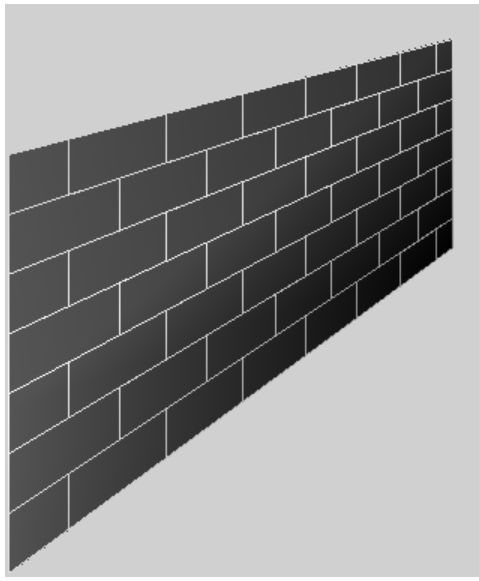

Abbildung 6.10: Perspektivisch gezeichnete Ziegelsteinmauer (siehe auch Farbteil, Abbildung F.6)

6.4 Zusammenfassung

- Zur Berechnung von perspektivischen geometrischen Figuren wird eine nicht lineare Transformation benötigt. GDI+ bietet hierfür die Methode ***Warp***, die für den ***Graphics Path*** definiert ist.
- Beleuchtungseffekte lassen sich mit Hilfe von ***Gradient Brushes*** simulieren. Benötigt man mehr als einen Farbverlauf, kann man den *Path Gradient Brush* einsetzen. Die Eigenschaft ***CenterPoint*** bestimmt die Lichtquelle, die Eigenschaften ***CenterColor*** und ***SurroundColors*** definieren die Farbverläufe zwischen dem *CenterPoint* und (in unserem Beispiel) den Eckpunkten der Mauer.
- Ein erweitertes Beispiel finden Sie auf der Website *www.dotnet-essentials.de*.