An Euclidean Vector's Application Example

A small case study to demonstrate the positive effect the usage of Euclidean vectors can have on software readability.

Andreas Harnack
ah8 at freenet dot de

November 2007

To demonstrate the effect the use of orientational analyses and Euclidean vectors can have, please consider the following two examples. Both examples are life code. The first one is a function taken from `konsole`, the terminal emulation that comes with Linux KDE[1]:

```
/* cited from [TEWidget.C] Terminal Emulation Widget */
/* version 1.0.2 */

void TEWidget::paintEvent( QPaintEvent* pe )
{
    const QPixmap* pm = backgroundPixmap();
    QPainter paint;
    setUpdatesEnabled(FALSE);
    paint.begin( this );
    paint.setBackgroundMode( TransparentMode );

    QRect rect = pe->rect().intersect(contentsRect());

    QPoint tL  = contentsRect().topLeft();
    int    tLx = tL.x();
    int    tLy = tL.y();

    int lux = QMIN(columns-1, QMAX(0,(rect.left() - tLx - blX ) / font_w));
    int luy = QMIN(lines-1, QMAX(0,(rect.top() - tLy - bY ) / font_h));
    int rlx = QMIN(columns-1, QMAX(0,(rect.right() - tLx - blX ) / font_w));
    int rly = QMIN(lines-1, QMAX(0,(rect.bottom() - tLy - bY ) / font_h));

    QChar *disstrU = new QChar[columns];
    for (int y = luy; y <= rly; y++)
    for (int x = lux; x <= rlx; x++)
    {
        int len = 1;
        disstrU[0] = fontMap(image[loc(x,y)].c);
        Attribute a(image[loc(x,y)]);
        while (x+len <= rlx && image[loc(len,y)] == a )
        {
            disstrU[len] = fontMap(image[loc(x+len,y)].c);
            len += 1;
        }
        QString unistr(disstrU,len);
        drawAttrStr(paint,
                QRect(blX+tLx+font_w*x,bY+tLy+font_h*y,font_w*len,font_h),
                unistr, image[loc(x,y)], pm != NULL, false);
        x += len - 1;
    }
    delete [] disstrU;
    drawFrame( &paint );
    paint.end();
    setUpdatesEnabled(TRUE);
}
```

The second example is the corresponding function providing exactly the same functionality, taken from a rewrite of a similar application[2]:

```
void qtWidget::paintEvent( QPaintEvent* e )
{
    BLOCK(qtWidget::paintEvent( QPaintEvent* e ))

    QPainter paint;
    paint.begin(this);
    setUpdatesEnabled(FALSE);
    cursor.hide(this, paint);

    rect a = intersect(rectangle(e->rect()), area()) - orig();
    rect r = rect(a.p0(), a.p1()+font.size()-vect(1)) / matrix(font.size());

    for ( dimY y = dimY(r.p0()); y < dimY(r.p1()); ++y) {
        CONSTRAIN( y < image.lines() )
        for ( dimX x = dimX(r.p0()); x < dimX(r.p1()); ) {
            CONSTRAIN( x < image.columns() )
            Attribute a(image[(x,y)]);  // operator[] takes a vector created by (,)
            dimX len(x+dimX(1));
            str[x()] = image[(x,y)];
            while ( len < dimX(r.p1()) && image[(len,y)] == a )
                str[len()] = image[(len,y)], ++len;
            drawString(paint, (x,y), str+x(), len-x, a);
            x = len;
        }
    }

    cursor.draw(this, paint);
    drawFrame( &paint );
    setUpdatesEnabled(TRUE);
    paint.end();
}
```

I think it's apparent that the code becomes more compact, better structured, better readable and, first of all, saver; in particular the bits providing the layout calculations.[3]

[1] This is an older version, published in 2001, the current version might look differently. However, that is the version I used to work with a couple of years ago, when I used the code in one of my own projects. For the sake of comparison, I removed all comments and replaced code that affects readability but is unrelated to vectors or dimensions; this code is printed in italic font.

[2] Comments have been removed too.

[3] By having chosen that particular example I do not mean to assess or even criticise the quality of the code written by someone else. I've been using the application and reusing the code for years. I present it here merely as an example to demonstrate the effect a programming technique can have on the code structure. I could have chosen any other piece of code.