Widgets translucides dans Qt

Qt by Nokia

par Geir Vattekar traducteur : Louis du Verdier Qt Quarterly

Date de publication : 29/08/2009

Dernière mise à jour : 01/08/2011

Une fonctionnalité nouvelle et non présentée de Qt 4.5 est les widgets translucides de haut niveau fenêtres dans lesquelles on peut voir au travers. Les applications utilisant la translucidité beaucoup de lecteurs média, par exemple ont constitué une vue courante des bureaux d'aujourd'hui, et les demandes au sujet de ce dispositif sont tout à fait communes sur les listes d'adresses de Qt. Contre toute attente, il n'y a pas beaucoup de kits d'utilitaires à l'extérieur qui supportent la translucidité de manière multiplateforme, nous avons donc vu le besoin d'exposer cette fonctionnalité par le biais de l'écriture de cet article.

Le processus d'application d'effets de translucidité aux widgets de Qt est banal : utilisez simplement avec un **QPainter** des couleurs avec un canal alpha durant la peinture des widgets. Si vous souhaitez que le widget entier ait le même degré de translucidité, nous avons une très bonne fonction de commodité, qui enlève le problème du dessin du widget. Dans cet article, nous allons explorer les possibilités que nous fourni Qt par l'implémentation d'un lecteur média (c'est-à-dire un frontal vide et silencieux d'un lecteur média) ainsi qu'une horloge analogique.

Cet article est une traduction autorisée de **Translucent widgets**.

I - L'article original

Qt Quarterly est une revue trimestrielle électronique proposée par Nokia à destination des développeurs et utilisateurs de Qt. Vous pouvez trouver les **versions originales**.

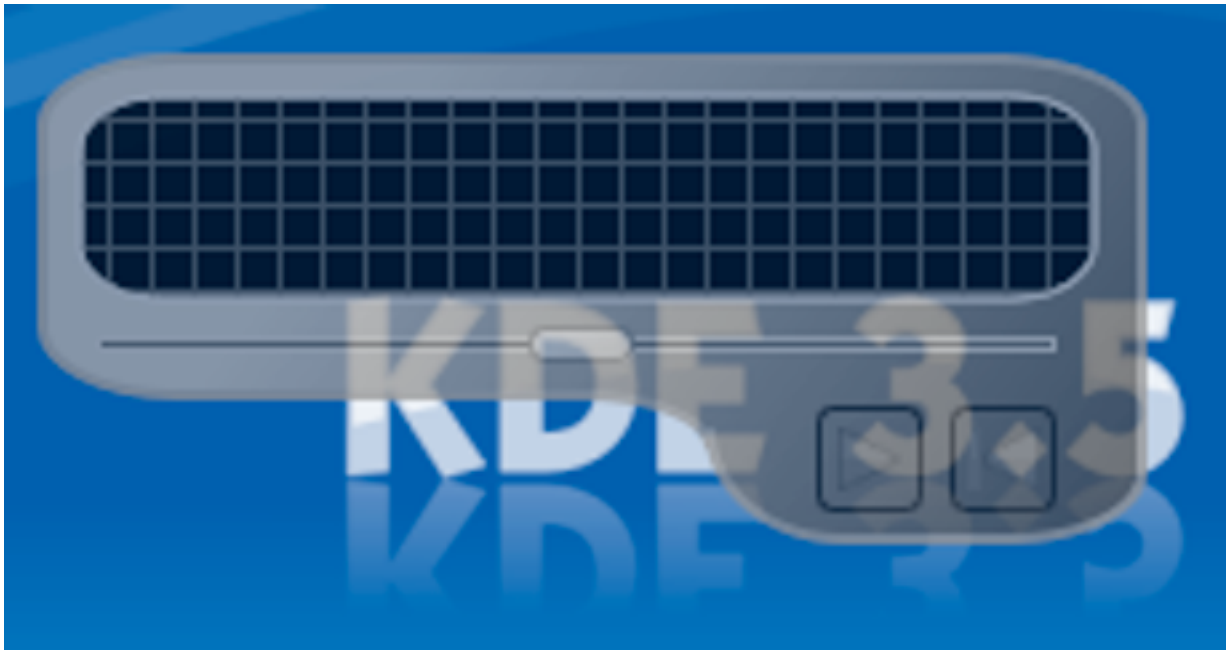
Cet article est la traduction de l'article **Translucent Widgets in Qt** paru dans la **Qt Quarterly Issue 29**.

Cet article est une traduction d'un des tutoriels écrits par **Nokia Corporation and/or its subsidiary(-ies)** inclus dans la documentation de Qt, en anglais. Les éventuels problèmes résultant d'une mauvaise traduction ne sont pas imputables à Nokia.

II - Un lecteur média translucide

Dans ce premier exemple, nous laissons le widget, le lecteur média, garder le même degré de translucidité. Comme nous allons le voir plus tard, Qt supporte aussi les effets de translucidité par pixel.

Commençons en donnant une image du lecteur afin de montrer ce que nous voulons réaliser :



Lecteur média

Lorsque la souris est au-dessus du lecteur média, nous le rendons opaque. Lorsque la souris quitte le lecteur, nous le réglons de manière à ce qu'il soit translucide. Par chance pour nous, Qt fournit une fonction de commodité pour parvenir à de tels effets. Ici, nous avons deux gestionnaires qui sont appelés lorsque la souris entre et quitte le lecteur.

```
void Player::enterEvent(QEvent *event)
{
    setWindowOpacity(1.0);
}
void Player::leaveEvent(QEvent *event)
{
    setWindowOpacity(0.5);
}
```

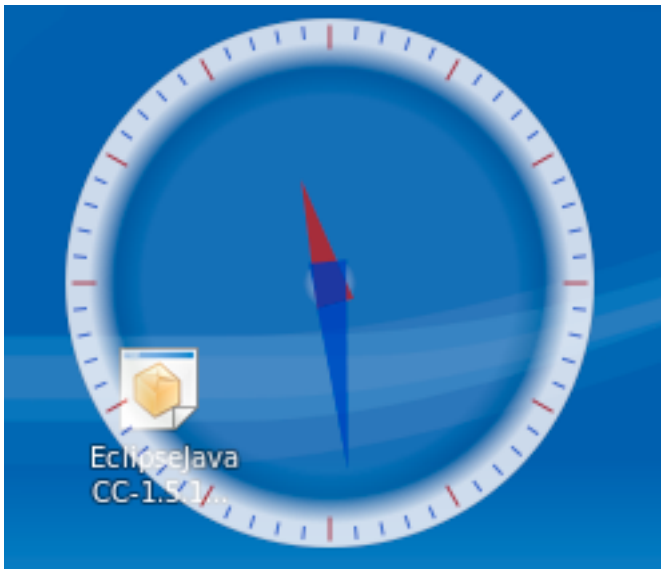
Les développeurs sous Windows 2000 et plus récent, sous Mac OS X et sous les plateformes modernes de X11 ont eu la capacité de changer l'opacité d'une fenêtre entière pendant quelques temps. Cet effet est utile mais limité, particulièrement si nous avons besoin de changer l'opacité de régions individuelles ou de pixels.

Les limitations de cet effet sont soulignées dans le cas où nous voudrions appliquer un masque au widget pour réaliser une fenêtre de forme non usuelle voir l'exemple **Shaped Clock** pour plus de détails. Les fenêtres de forme particulière créées de cette manière ont souvent les contours déchiquetés puisqu'il n'existe aucune façon de joindre leurs bordures avec leurs environnements.

Ce que nous voulons vraiment est de pouvoir former des pixels individuels au contour de la fenêtre translucide tout en maintenant les autres parties de la fenêtre opaques. L'exemple qui suit montre comment cela peut être fait.

III - Une horloge analogique translucide

Dans cet exemple, nous peignons le widget en utilisant des couleurs à canal alpha. Nous retournons maintenant à la familière horloge analogique trouvée dans les exemples standards de Qt. Mais nous allons l'étoffer un peu en rendant ses bordures transparentes et en donnant à son centre un joli effet de translucidité. Vous pouvez voir ci-dessous une image de l'horloge ci-dessous.



Lecteur média

Regardons le fichier d'en-tête de notre horloge transparente :

```
class Clock : public QWidget
{
    Q_OBJECT
public:
    Clock(QWidget *parent = 0);
    QSize sizeHint() const;
protected:
    void mouseMoveEvent(QMouseEvent *event);
    void mousePressEvent(QMouseEvent *event);
    void paintEvent(QPaintEvent *event);
private:
    QPoint dragPosition;
};
```

Depuis que nous n'avons pas de cadre de fenêtre, nous rendons possible de déplacer l'horloge en la saisissant à la souris. Nous implémentons cela dans **mouseMoveEvent()** et **mousePressEvent()**. dragPosition garde une trace de l'endroit où le lecteur est situé sur l'écran.

Mais c'est la peinture translucide qui nous intéresse. Nous faisons tout cela dans la fonction **paintEvent()**. Cependant, avant que nous puissions commencer à peindre, nous allons avoir besoin de mettre en marche la translucidité. Le code qui fait cela se situe dans le constructeur.

```
Clock::Clock(QWidget *parent)
    : QWidget(parent, Qt::FramelessWindowHint | Qt::WindowSystemMenuHint)
{
    ...
    setAttribute(Qt::WA_TranslucentBackground);
    ...
    setWindowTitle(tr("Analog Clock"));
}
```

Définir le drapeau **Qt::WA_TranslucentBackground** représente tout ce qui est requis avant que nous puissions commencer à peindre. De plus, les zones dont nous n'allons pas peindre l'intérieur deviendront entièrement transparentes (à moins que nous décidions de peindre l'arrière-plan dans une couleur opaque, bien sûr). Notez que nous réglons l'indication de fenêtre **Qt::FramelessWindowHint** afin d'obtenir une fenêtre sans cadre.

Passons au code de peinture :

```
void Clock::paintEvent(QPaintEvent *)
{
    ...
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);
```

La première chose que nous faisons est de mettre la méthode de rendu anti-aliasing. Comme mentionné, cela donne à l'horloge un contour joli et lisse. Puis nous commençons la peinture :

```
QRadialGradient gradient(0.0, 0.0, side*0.5, 0.0, 0.0);
    gradient.setColorAt(0.0, QColor(255, 255, 255, 255));
    gradient.setColorAt(0.1, QColor(255, 255, 255, 31));
    gradient.setColorAt(0.7, QColor(255, 255, 255, 31));
    gradient.setColorAt(0.8, QColor(0, 31, 0, 31));
    gradient.setColorAt(0.9, QColor(255, 255, 255, 255));
    gradient.setColorAt(1.0, QColor(255, 255, 255, 255));
    painter.setPen(QColor(0, 0, 0, 32));
    painter.setBrush(gradient);
    painter.drawEllipse(-side/2.0 + 1, -side/2.0 + 1, side - 2, side - 2);
    ...
```

Comme vous pouvez le voir, nous n'avons pas à faire quoique ce soit de particulier avec le peintre. Après avoir défini l'attribut **Qt::WA_TranslucentBackground**, nous avons juste peint comme d'habitude. Ici, nous dessinons le cercle interne de l'horloge en utilisant un gradient radial coloré de manière translucide. C'est cela qui donne l'effet à l'intérieur de l'horloge, comme vous pouvez le voir sur la capture d'écran.

Nous n'allons pas vous ennuyer avec une explication complète de la fonction **paintEvent()**. Il suffit de dire ceci : nous dessinons le reste de l'horloge ; c'est-à-dire, les deux aiguilles, et les marqueurs d'heure et de minutes sur la bande.

IV - Le serveur X et les gestionnaires de fenêtres sous Linux

Sous Windows 2000 (et versions suivantes) et sous Mac OS X, la translucidité et la transparence devraient fonctionner à l'extérieur du cadre. Sous Linux, quelques mises au point peuvent être requises comme cela est souvent le cas pour des effets graphiques de pointe (et souvent aussi certains effets pas si sophistiqués). Notamment, le serveur X nécessite de supporter les valeurs de pixels ARGB et un gestionnaire de fenêtres composite est requis.

Nous ne pouvons entrer dans les détails pour toutes les distributions disponibles et pour tous les gestionnaires de fenêtrage, principalement parce que cela prendrait beaucoup trop de place dans ce petit article (et que cela n'a pas été testé sur la totalité d'elles). Une rapide recherche Internet vous donnera les informations dont vous avez besoin. Comme exemple, nous pouvons mentionner que, sous Kubuntu avec KDE 3, nous avons besoin de mettre

explicitement la translucidité en marche comme cela est considéré expérimentalement. Avec le nouveau KDE 4, cela est mis en marche par défaut ; en fait, plusieurs effets de bureau qui viennent avec KDE 4 utilisent les effets de translucidité.

V - Fin ?

Il y a de nombreuses possibilités comme il en vient pour les widgets translucides. Dans cet article, nous avons uniquement joué avec les widgets de haut niveau. Il est aussi de peindre des widgets enfants en utilisant des couleurs translucides. Cela peut par exemple être utilisé pour les vues d'éléments.

Comme mentionné, le lecteur média n'est actuellement qu'une application vide et quelque peu monotone. Nous prévoyons de l'étendre en utilisant le nouveau framework d'animation de Qt, dont la sortie est prévue dans Qt 4.6. Pour de bonnes mesures, nous pouvons aussi implémenter les fonctionnalités médiatiques en utilisant le framework multimédia Phonon. Restez donc à l'écoute pour tout développement ultérieur.

VI - Divers

J'adresse ici de chaleureux remerciements à **dourouc05** pour sa patience et son aide.

Au nom de toute l'équipe Qt, j'aimerais adresser le plus grand remerciement à Nokia pour nous avoir autorisé la traduction de cet article !