Qt Graphics et performances : le moteur de rendu OpenVG

Par rweather - Qt Labs - Thibaut Cuvelier (traducteur)

Date de publication : 11 octobre 2010

Dernière mise à jour : 10 février 2020

Dans les précédents billets de cette série, Gunnar a décrit le design et les caractéristiques du système de dessin de Qt et a exploré le moteur Raster plus profondément. Dans cet article, je vais décrire les fonctionnalités uniques du système graphique OpenVG.

N'hésitez pas à commenter cet article !
Commentez

I - L'article original

Qt Labs est un site géré par les développeurs de Qt. Ils y publient des projets, des idées propres et des composants afin d'obtenir les retours d'informations sur les API, le code et les fonctionnalités ou simplement pour partager avec nous ce qui les intéresse. Le code que vous y trouverez peut fonctionner tel quel, mais c'est sans aucune garantie ni aucun support. Voir les **conditions d'utilisation** pour plus d'informations.

Cet article est la traduction de l'article **Qt Graphics and Performance – OpenVG** de **rweather** paru dans Qt Labs.

II - Le moteur de dessin

À l'inverse des autres moteurs, OpenVG était beaucoup plus facile à implémenter, car l'API OpenVG elle-même était fort proche des fonctionnalités de QPainter. Vous pouvez regarder les spécifications **sur le site de Khronos**, mais voici les points principaux :

- les objets VGPath représentent une géométrie à base d'éléments MoveTo, LineTo et CubicTo. Ceci est fort proche des couches d'abstraction QPainterPath et QVectorPath de Qt ;
- les objets VGPaint représentent des brosses et des pinceaux pour remplir les chemins avec des motifs de pixels. Des couleurs solides, des dégradés linéaires, radiaux et les motifs de brosses sont supportés, mais pas les dégradés coniques ;
- les objets VGImage représentent des pixmaps dans une large gamme de formats ; OpenVG supporte beaucoup plus de formats que OpenGL/ES, ce qui facilite la conversion d'une QImage en VGImage ;
- les objets VGFont (uniquement pour OpenVG 1.1) stockent des glyphes représentés sous forme de VGImage ou de VGPath pour un rendu plus rapide des textes ; sous OpenVG 1.0, nous nous replions sur le dessin de chemins;
- des ciseaux pour la découpe basée sur des rectangles ;
- un masque alpha pour la découpe de formes arbitraires ;
- des actrices de transformation affine pour le dessin de chemins et de glyphes, des matrices de transformation affine et projective pour le dessin d'images.

III - Matrices de transformation

OpenVG ne supporte pas les matrices de transformation projective pour le dessin de chemins, ce qui est ennuyant, car QPainter permet à n'importe quelle QTransform affine ou projective d'être utilisée pour n'importe quelle opération de dessin. Il y a une extension OpenVG enregistrée, **VG_NDS_projective_geometry**, mais aucun des moteurs OpenVG que nous avons trouvés ne la supporte. La raison ? Générer des pixels en perspective peut être assez difficile. Les matrices de projection sont supportées pour le dessin d'images, car dessiner une simple image en perspective est un problème bien compris que les systèmes OpenGL/ES résolvent tout le temps.

Quand une matrice de transformation projective est utilisée pour le dessin de chemins, nous convertissons ce chemin point par point en utilisant QTransform et la dessinons comme un chemin « normal » affine en utilisant une transformation par défaut pour la surface de la fenêtre. Quid des pixels peints ? Malheureusement, leur perspective ne sera pas correcte. En pratique, ce n'est pas un gros problème : la majorité des chemins sont dessinés avec une brosse de couleur solide et les couleurs solides sont parfaitement semblables à elles-mêmes en perspective.

En général, cependant, nous décourageons les gens d'utiliser des transformations projectives avec des chemins. Si vous voulez réellement dessiner une scène en perspective, dessinez-la d'abord dans un QPixmap et ensuite dessinez le pixmap en utilisant une transformation projective. Vous auriez de toute façon dû faire ainsi parce que les transformations ont principalement lieu pendant les effets d'animation de « feuilletage » - dessiner chaque petit chemin en perspective à chaque frame lors de cette animation serait trop lent.

IV - Transformation de chemin et dessin

La majorité de la logique de transformation de chemin est effectuée dans vectorPathToVGPath() et painterPathToVGPath() du fichier qpaintengine_vg.cpp. Nous détectons la présence de matrices de transformation affine ou projective et utilisons une conversion appropriée. Nous convertissons QVectorPath et QPainterPath à l'aide de routines spécialisées. Les autres moteurs de rendu convertissent, habituellement, tout dans un QVectorPath d'abord. La conversion en QPainterPath peut légèrement augmenter les performances quand des chemins arbitraires sont dessinés pendant le rendu SVG et autres opérations du même acabit - il est inutile de créer un QVectorPath s'il va être rapidement jeté.

Le rendu de chemins utilise une approche de mise à jour paresseuse, tentant de minimiser le nombre de changements d'état OpenVG de requête à requête :

- si le dessin requiert un stylo, l'objet penPaint est mis à jour avec le QPen actuel s'il était différent de celui utilisé la dernière fois ;
- si le dessin requiert une brosse, l'objet brushPaint est mis à jour avec la QBrush actuelle si elle était différente de celle utilisée la dernière fois ;
- la matrice de transformation de chemin est mise à jour si elle a changé depuis la dernière opération de dessin de chemin ;
- le chemin est dessiné en utilisant vgDrawPath().

La majorité de l'état OpenVG persiste entre les événements de dessin ; ainsi, si le même stylo est utilisé d'une frame à l'autre, il ne sera défini qu'une fois et ne changera jamais. L'état est aussi partagé entre toutes les fenêtres, parce qu'il n'y a qu'un seul contexte OpenVG pour l'entièreté du système.

Dans une version précédente du moteur de dessin OpenVG, je ne faisais que reporter les changements d'état dès qu'ils apparaissaient, sans tenter d'être plus paresseux à ce sujet. C'était une erreur.
Les applications qui utilisent QPainter, particulièrement celles qui utilisent QGraphicsView, peuvent être très causantes, à constamment sauver et restaurer l'état du painter. Il était assez commun de changer les brosses, les stylos et les matrices de transformation, de les changer à nouveau... sans rien dessiner. Maintenant, il ne met à jour l'état OpenVG qu'au moment où une opération de dessin est sur le point de se produire. Cette tâche ménagère a cependant un coût ; ainsi, si vous pouvez éviter de changer l'état d'un QPainter dans votre application, faites-le.

V - Chemins préalloués

Rectangles, lignes, points et rectangles arrondis apparaissent assez fréquemment dans beaucoup d'applications, avec de constants changements de coordonnées. OpenVG nous fait créer et détruire un VGPath à chaque fois. Pour l'éviter, nous avons proposé des chemins préalloués pour les opérations simples de dessin, que nous mettons à jour avec vgModifyPathCoords() au lieu d'allouer de la mémoire sur le GPU pour un nouveau chemin. Cependant, certains chipsets peuvent être plus lents à la modification d'un chemin qu'à sa création ! Sur ceux-là, compilez Qt avec la macro QVG_NO_MODIFY_PATH.

VI - Rendu d'image

Le meilleur dessin d'image sera effectué avec QPixmap au lieu de QImage. En effet, avec QPixmap, l'image est convertie en VGImage une fois et puis dessinée plusieurs fois. Avec QImage, l'image doit être convertie en VGImage à chaque fois qu'elle est dessinée.

La primitive de dessin OpenVG vgDrawImage() est très primitive - elle dessine la VGImage sélectionnée à l'origine de la transformation actuelle. Il n'y a pas de support intégré de dessin dans des sous-rectangles. Heureusement, OpenVG dispose de vgChildImage(), qui rend très rapide l'extraction d'une sous-région, les données des pixels étant partagées avec le parent. Cependant, « rapide » est un terme très relatif - j'ai vu des cadences d'image réduites de moitié en utilisant vgChildImage() en comparaison du dessin d'une image complète. Ainsi, si cela est possible dans votre cas, dessinez des QPixmap entières avec OpenVG et limitez l'utilisation des sous-rectangles.

Une autre source de ralentissement est le dessin d'images avec opacité. OpenVG dispose d'une méthode pour multiplier un objet VGPaint avec une VGImage pour produire une image de destination. C'est une méthode très peu coûteuse d'obtention d'effets d'opacité et assez rapide.
Sauf ! Il y a toujours un « sauf » : quand l'image est dessinée avec une matrice de transformation projective...
Rappelez-vous - des pixels peints ne peuvent pas être générés en perspective -, nous ne pouvons donc pas utiliser un objet de dessin pour générer une « couleur d'opacité », même si la couleur d'opacité solide est la même en perspective pour tous les angles ! Ceci est très ennuyant - le comité OpenVG aurait pu faire une exception spéciale pour les objets VGPaint de couleur solide.

Quand le moteur de dessin OpenVG dessine une image avec une certaine opacité et qu'une transformation projective est utilisée, nous devons générer une copie du VGImage et utiliser vgColorMatrix() pour ajuster l'opacité. Ceci n'est pas trop mauvais si vous dessinez toujours la même image encore et toujours avec la même opacité, mais c'est extrêmement inefficace si vous animez l'opacité. Évitez ainsi les animations d'opacité avec OpenVG si vous le pouvez.

Dessiner sur un QPixmap n'est pas actuellement accéléré avec OpenVG - cela utilise plutôt le moteur de dessin *raster*, nous recommandons donc de peindre des pixmaps une fois, plutôt que de les mettre à jour. Nous résoudrons ce problème dans des versions futures. Même quand nous implémentons le dessin sur des pixmaps, il y aura un certain coût : passer d'une surface de rendu à une autre, d'une fenêtre à une VGImage n'est pas gratuit - sur certains chipsets, cela peut être aussi lourd que le changement complet de contexte EGL. Évitez ainsi de changer de surface de dessin si possible.

VII - Découpe

La découpe est l'un des fléaux de mon existence ! Cela semble si facile aux auteurs d'applications - définir un rectangle de découpe et ce sera efficace, en dessinant moins de pixels. Si seulement !

Il y a trois techniques qui peuvent être utilisées pour la découpe avec OpenVG :

- la liste de rectangles de découpe ;
- le masque alpha pour des formes de découpe arbitraires ;
- le rectangle de découpe pour des découpes simples et des masques alpha pour des découpes plus complexes.

La dernière est celle sélectionnée par défaut dans le moteur de dessin OpenVG et des #define existent pour activer les autres modes. Cependant, sur certaines versions des chipsets PowerVR, il y a un bogue qui fait que, si le ciseau est combiné à un masque alpha, les performances chutent fortement - jusque deux frames par seconde dans certains cas ! Sur de tels périphériques, vous pourriez vouloir activer soit la découpe, soit le masque.

Le mieux est de ne pas utiliser du tout la découpe si c'est possible. Dessinez tout dans votre scène et laissez le GPU faire le lifting. Souvenez-vous que les GPU modernes OpenGL/ES peuvent remonter des milliers de triangles par seconde, avec des algorithmes intelligemment pensés pour la suppression des faces cachées, qui sont bien plus intelligents que tout ce que vous pouvez faire avec des découpes. OpenVG utilise le même GPU dans la majorité des cas. Si vous spécifiez une découpe, vous pourriez confondre le GPU et lui faire emprunter un chemin interne plus lent que ce qu'il aurait fait.

Si vous devez découper, essayez d'utiliser des régions en rectangles simples qui peuvent être définis par le ciseau.

VIII - Surfaces de fenêtres

En dessous du moteur de dessin OpenVG, il y a la logique de surface de fenêtre dans le système graphique. Généralement, des personnalisations spécifiques à la plate-forme sont requises ici pour que les pixels s'affichent aussi vite que possible. La classe QVGWindowSurface emballe un objet QVGEGLWindowSurfacePrivate, qui fournit tout le traitement lourd. L'implémentation par défaut de EFL est QVGEGLWindowSurfaceDirect, qui écrit des pixels dans le tampon de la fenêtre et appelle eglSwapBuffers() pour le transférer à l'écran. C'est possible d'activer des

opérations avec un seul tampon avec QVG_DIRECT_TO_WINDOW mais le coût pourrait montrer quelques artéfacts à l'écran.

Si votre plate-forme dispose de quelques mécanismes intelligents d'extension EFL pour mettre des pixels à l'écran, vous devrez écrire un nouveau plug-in de système graphique et implémenter votre propre sous-classe de QVGEGLWindowSurfacePrivate. Le module QtOpenVG a été structuré pour rendre cela relativement facile sans devoir toucher au code du cœur Qt.

IX - Utilisation de la mémoire

Tout ce que vous faites utilise de la mémoire - et dans le CPU, et dans le GPU. Les surfaces de fenêtre, les contextes de rendu VG, les objets VGPath, les objets VGImage et tous les autres. Ce qui peut devenir un peu serré sur un GPU dans des systèmes embarqués. Nous avons pris quelques respirations pour gérer ceci : par exemple, détruire les objets VGImage précédents lors d'un téléversement d'un nouveau QPixmap, détruire tous les objets OpenVG quand une application part en arrière-plan pour libérer la mémoire pour des applications à l'avant-plan.

Plus vos applications sont complexes, plus elles vont probablement heurter de plein fouet la limite de la mémoire du GPU. Peu de choses, voici à quoi se résume l'utilité du module QtOpenVG dans de pareils cas. Au mieux, il est possible de prendre quelques mesures d'urgence pour se récupérer... mais c'est tout. Gardez donc à l'œil le nombre de pixmaps et de fenêtres que vous utilisez et voyez si vous pouvez simplifier un tant soit peu votre application. Par contre, n'envoyez pas de grosse photographie JPEG en un seul QPixmap - divisez-la en plusieurs tuiles (« tiles ») qui peuvent être nettoyées dès que la mémoire du GPU se remplit.

X - Résumé

Les conseils qui suivent résument les suggestions de performances de la section précédente :

- évitez les matrices de transformation projective lors du dessin de chemins ;
- minimisez les changements d'état sur les pinceaux, brosses, transformations, etc. ;
- utilisez QPixmap au lieu de QImage quand cela s'avère possible ;
- évitez de dessiner des images en utilisant des sous-rectangles ;
- lors du dessin d'images avec opacité, utilisez une matrice de transformation affine ou un seul niveau d'opacité ;
- évitez de changer de surface de dessin, particulièrement entre fenêtres et pixmaps ;
- n'utilisez pas de découpe si vous dessinez votre scène dans un ordre *bottom-up* ;
- découpez les grandes images en tuiles plus petites pour éviter de surcharger la mémoire du GPU.

XI - Et après ?

Il y a toujours plus à faire pour améliorer tout système logiciel. QtOpenVG ne fait pas exception :

- dessin dans des QPixmap en utilisant OpenVG ;
- regroupement plus intelligent des VGImage pour faire face aux conditions de la mémoire du GPU ;
- pilotes d'écran pour Qt/Embedded et Lighthouse.

XII - Divers

Merci à **sysedit**, **Mahefasoa** et **jacques_jean** pour leur relecture attentive !

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