Accélération matérielle dans Xorg et projet NouVeau

Stéphane Marchesin
marchesin@icps.u-strasbg.fr

Plan

- Introduction
- Description du matériel
- Ressources disponibles
- Accélération 2D
- Accélération 3D
- Ingénierie inverse
- Travaux effectués/en cours/futurs
- Conclusion

Introduction

- Nécessité de la 3D
  - Les systèmes de fenêtrage utilisent OpenGL
  - Applications 3D
  - Jeux

Introduction

- Problèmes avec les pilotes propriétaires
  - Requiert une architecture x86 ou x86_64
    - ➔ Pas de support PPC
    - ➔ Support IA64 quasi inexistant
  - Impossible à debugger
  - Support sur le long terme ?
    - ➔ Introduction de pilotes “legacy”
    - ➔ Pas de nouvelles fonctionnalités

Introduction

- Projet NouVeau
  - Basé sur le pilote libre “nv”
  - Amélioration du pilote pour la 2D
  - Pilote libre pour la 3D
  - Support des architectures exotiques
  - Support des cartes plus anciennes

Le matériel

- Plusieurs manières de communiquer avec un périphérique
  - Les registres (MMIO)
    - → Un tableau de valeurs
    - → Chaque case a une fonction
      - ex : couleur du triangle
      - ex : position du sommet
    - → Lent
    - → Pour changer la couleur d'un triangle
      - écrire la couleur du triangle dans la case correspondante
  - Les FIFOs
    - → Un bloc de mémoire écrit par le processeur et lu par la carte
    - → Rapide
    - → Pour changer la couleur d'un triangle
      - écrire deux valeurs dans la fifo : la commande “changer de couleur” puis la couleur elle même
      - avancer le pointeur de la fifo

Le matériel

- Riva 128 (NV03)
  - Support de plusieurs contextes en simultané
    - ➔ Sous la forme de plusieurs buffers de commandes
    - ➔ 8 contextes disponibles
  - OpenGL 1.1
  - Gestion de la mémoire
    - ➔ Notion d'objet
  - Documentation disponible
    - ➔ 2D documentée

Le matériel

- TNT (NV04)
  - Support de plusieurs contextes en simultané
    - ➔ 16 contextes disponibles
  - Refonte majeure du NV03
  - OpenGL 1.1
  - Documentation disponible
    - ➔ Processeur complètement documenté, 2D et 3D

Le matériel

- Geforce (NV10)
  - Support de plusieurs contextes en simultané
    - ➔ 32 contextes disponibles
  - Introduction du TCL matériel
  - OpenGL 1.2
  - Documentation disponible
    - ➔ Partielle, couvre une partie de la 2D seulement

Le matériel

- Geforce 3 (NV20)
  - Support de plusieurs contextes en simultané
    - ➔ 32 contextes disponibles
  - OpenGL 1.4
  - Documentation disponible
    - ➔ Aucune

Le matériel

- Geforce 6800 (NV40)
  - Support de plusieurs contextes en simultané
    - ➔ 32 contextes disponibles
  - OpenGL 2.0
    - ➔ Très puissant
    - ➔ Très complexe
  - Documentation disponible
    - ➔ Aucune

Ressources disponibles

- Code source
  - nvsdk, le SDK officiel de Nvidia
    - ➔ Pour les cartes TNT
    - ➔ Passé au préprocesseur
  - Le pilote UtahGLX (et le pilote haiku)
    - ➔ Fonctionne sur NV04 -> NV18
      - TNT, TNT2, Geforce 1, Geforce 2, Geforce 4 MX
    - ➔ Utilise les NV1x comme si c'étaient des TNT
      - Mode de compatibilité
- Documentation
  - Les en-têtes contenant les descriptions des registres
  - C'est tout

Accélération 2D

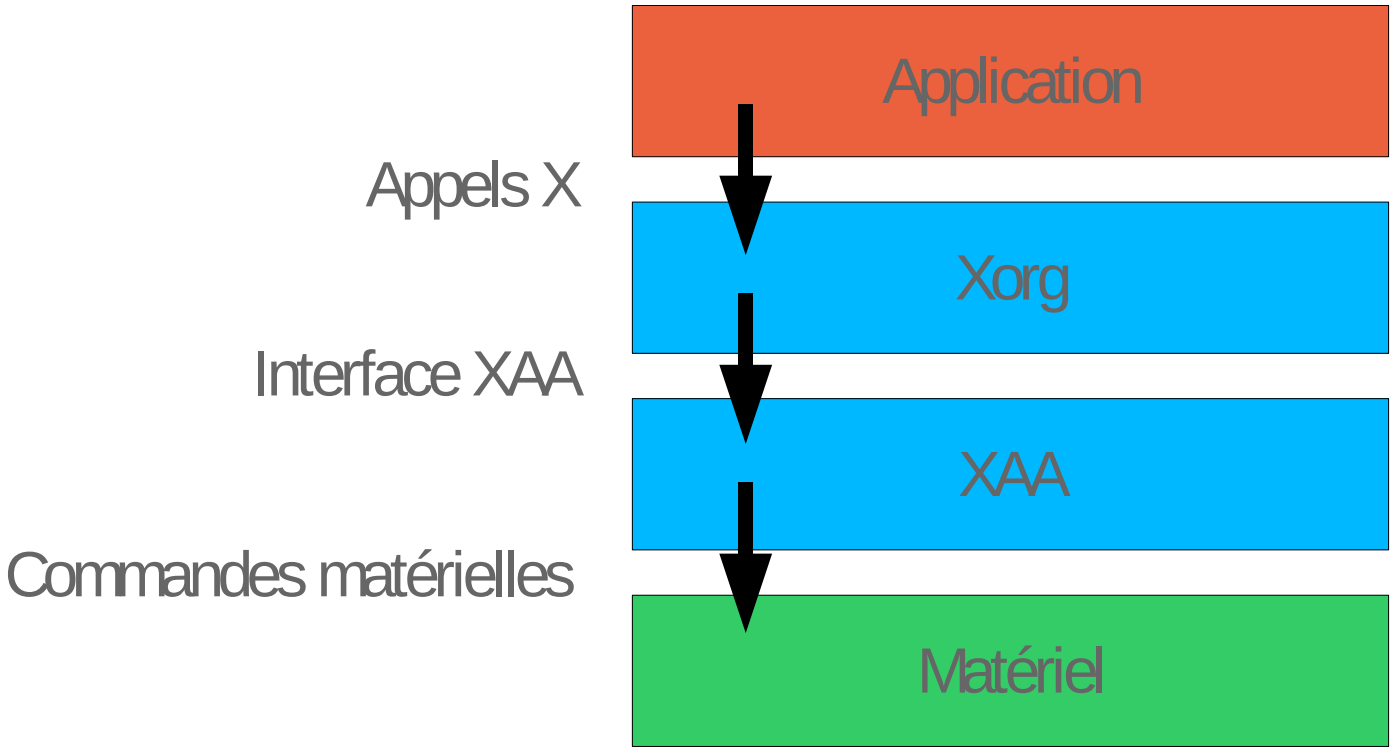
- Problématique
  - Stocker des pixmaps (images) en ram video
  - Faire des copies entre pixmaps
    - ➔ Copies opaques
    - ➔ Copies transparentes
  - La performance est secondaire
    - ➔ La vitesse est grandement limitée par les transferts de pixmaps sur le bus

Accélération 2D

- XAA
  - Plus ancien système d'accélération 2D
  - Accélère des primitives diverses
    - Remplissage de rectangles
    - Copie d'images opaques
    - Lignes
    - Lignes pointillées

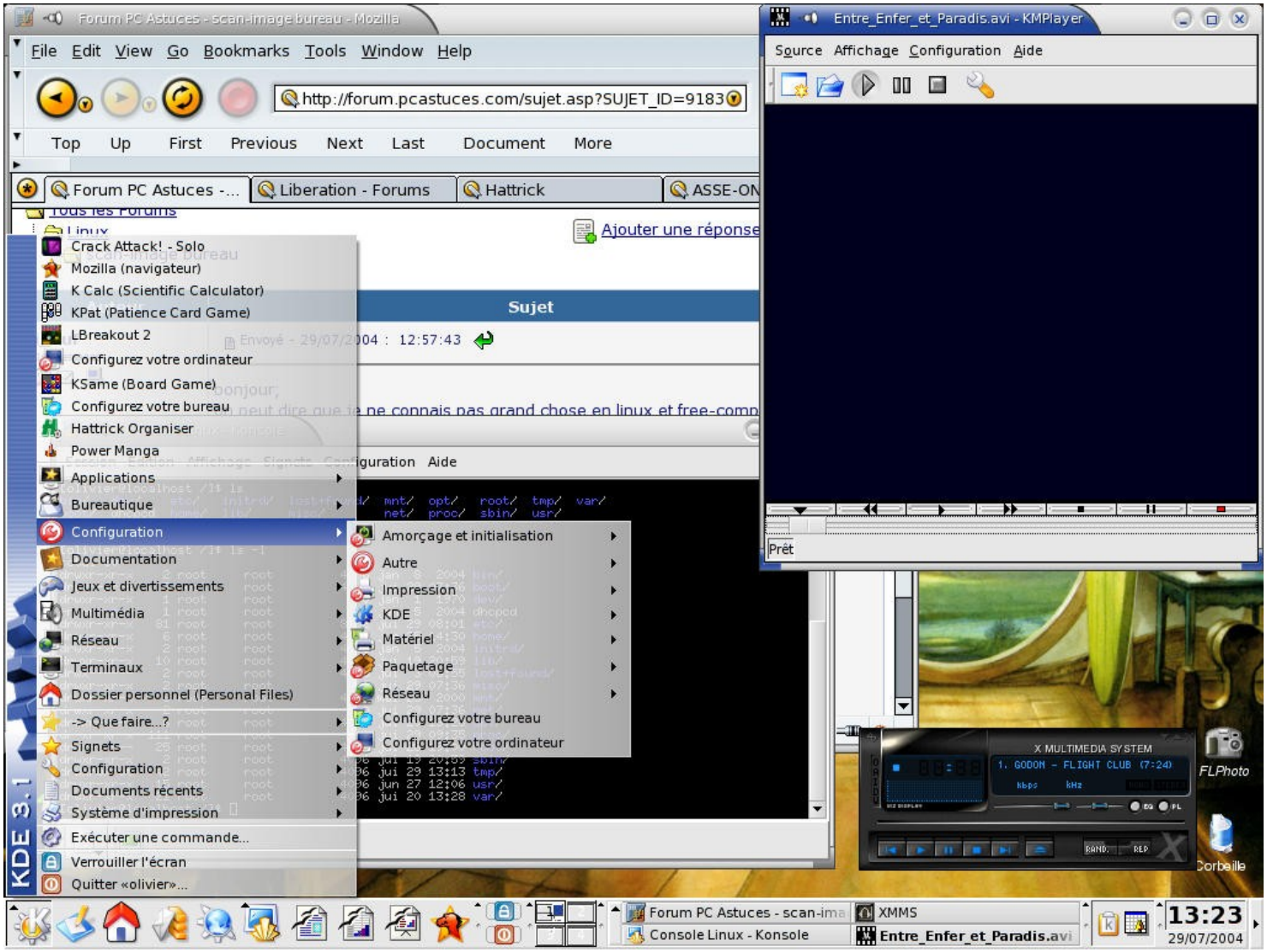
Accélération 2D

- XAA



Accélération 2D

- Interlude : combien de fois la primitive ligne est-elle utilisée dans cette image ?

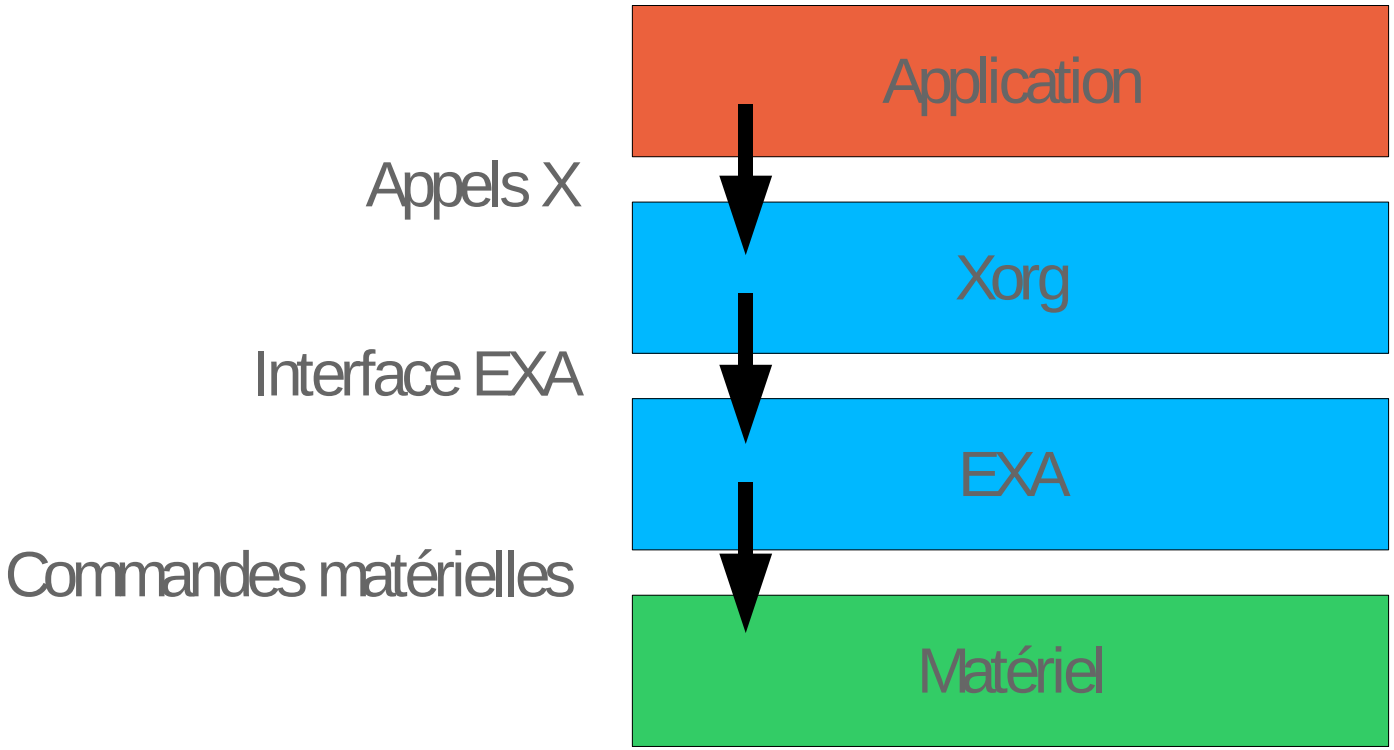


Accélération 2D

- EXA
  - Construit sur les problèmes de XAA
  - Accélère des primitives bien précises
    - ➔ Copie de zones opaques
    - ➔ Copie de zones avec transparence
    - ➔ Plus généralement, tous les opérateurs de Porter & Duff
    - ➔ Non, pas de lignes

Accélération 2D

- EXA



Accélération 2D

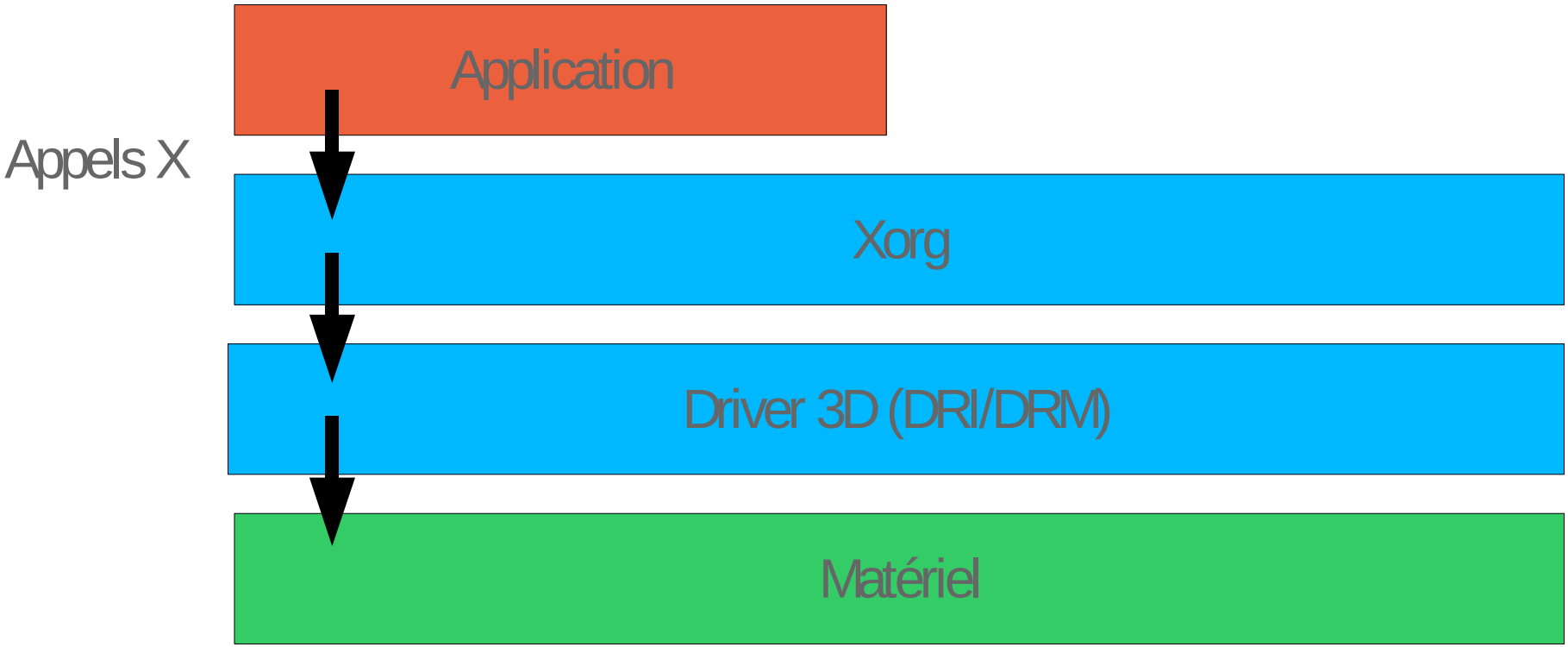
- EXA dans NouVeau
  - Implémente les fonctions de copie
  - Implémente les fonctions de transparence alpha
  - Permet les ombres et les transparences
  - Au moins aussi rapide que “nv”
  - EXA est utilisé par défaut dans NouVeau

Accélération 2D

- Xgl & co
  - Idée : utiliser OpenGL pour tout y compris la 2D
  - Accélère des primitives 2D en les réimplémentant par OpenGL
  - Nécessite la disponibilité d'un pilote OpenGL
    - ➔ Fourni par un second serveur X (Xglx)
    - ➔ Fourni par un mini serveur X (Xegl)
  - Ralentit la 3D
    - ➔ La 3D doit passer par un buffer indirect

Accélération 2D

- Xgl & co

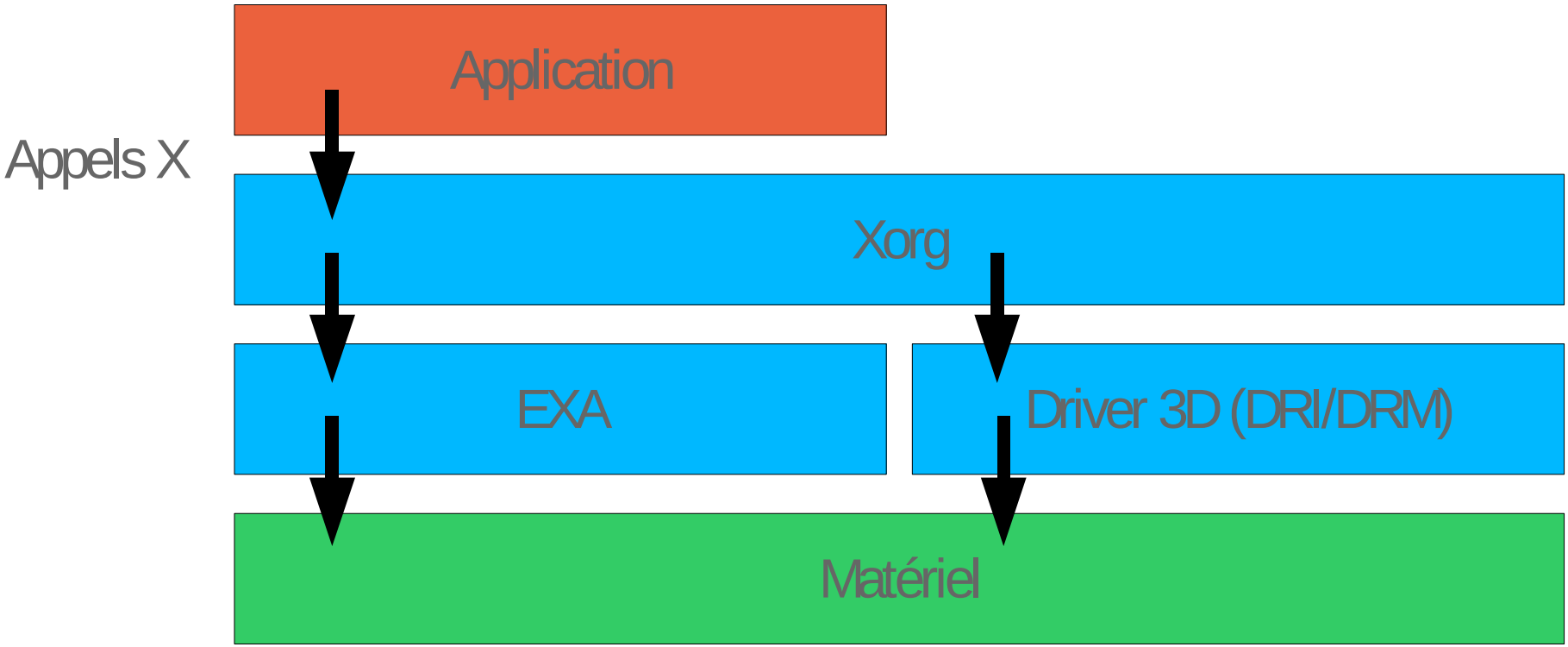


Accélération 2D

- AIGLX
  - Idée : garder la méthode d'accélération 2D “classique” (EXA)
  - Ajouter la possibilité d'utiliser des fonctions supplémentaires
    - ➔ Plus flexible
    - ➔ Ne ralentit la 3D que si on utilise les fonctions avancées

Accélération 2D

- AIGLX



Accélération 3D

- Problématique en 4 points :
  - Rendre des triangles
  - Rapidement
  - Rapidement
  - Rapidement

Accélération 3D : Le modèle DRI/DRM

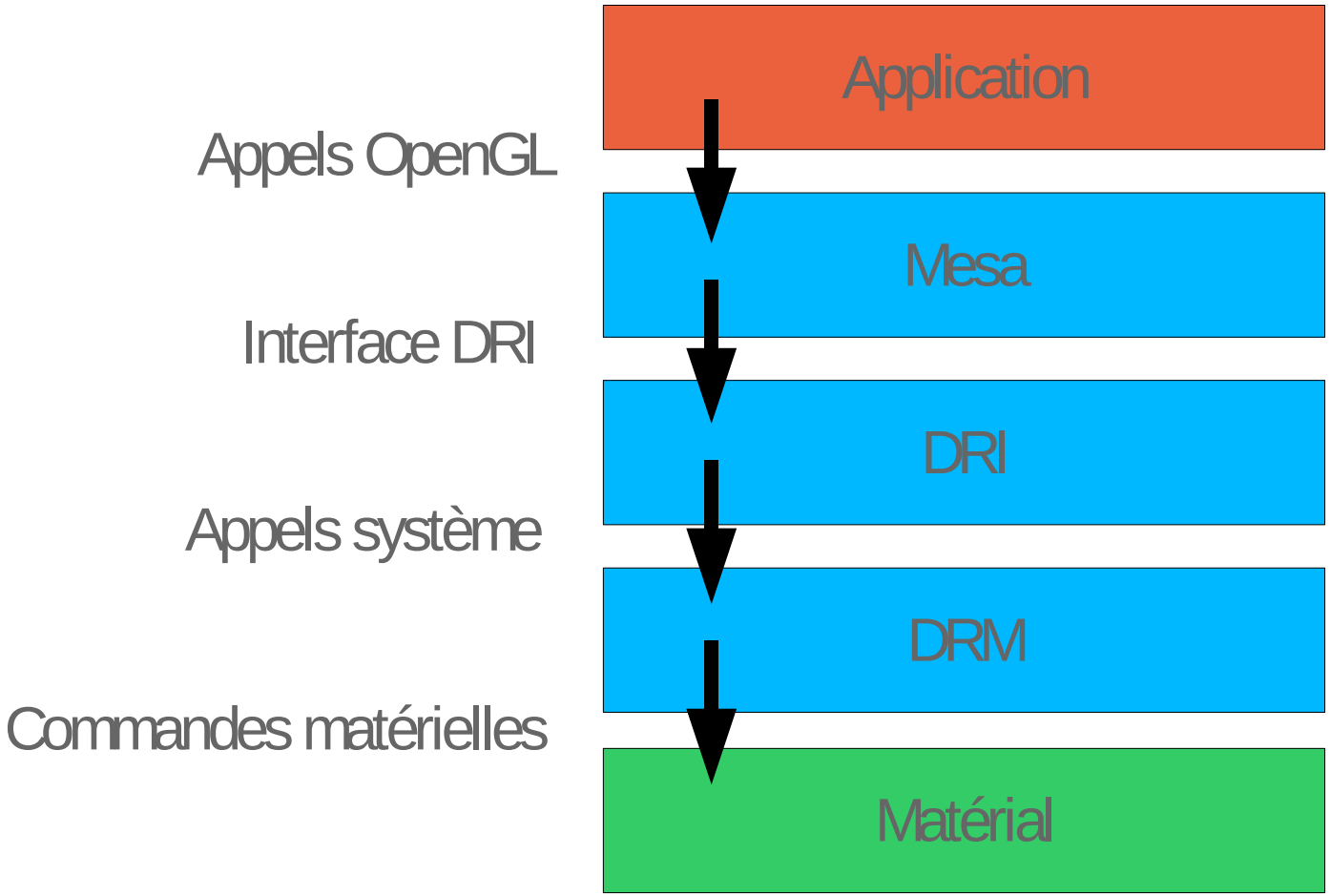
- Un pilote graphique ne peut pas se trouver dans le noyau
  - Compliqué à débugger
  - Risques de plantage d'un module noyau trop gros
- Idée : séparer le pilote en deux morceaux
  - Un module noyau (DRM)
    - ➔ Vérifie les accès à la carte faits par le DRI
      - Une carte graphique peut accéder à des zones quelconques de mémoire
      - Il est donc très facile de gagner les privilèges root si rien n'est fait
  - Un module tournant dans l'espace utilisateur (DRI)
    - ➔ Ce module n'est pas considéré comme sûr par le DRM

Accélération 3D : Le modèle DRI/DRM

- DRM : protège les accès au matériel
  - Dans le noyau
  - Taille minimale
  - Vérifie les commandes que le DRI envoie à la carte
    - ➔ Ralentit le rendu
    - ➔ Pour éviter trop de ralentissements, on garde une copie de l'état de la carte
      - Complexe
      - Terrain propice aux bugs
- DRI : fait la plupart du travail
  - Dans l'espace utilisateur
  - Est un plugin de Mesa
  - Construit des commandes qu'il soumet ensuite au DRM

Accélération 3D : Le modèle DRI/DRM

- Fonctionnement du DRI/DRM

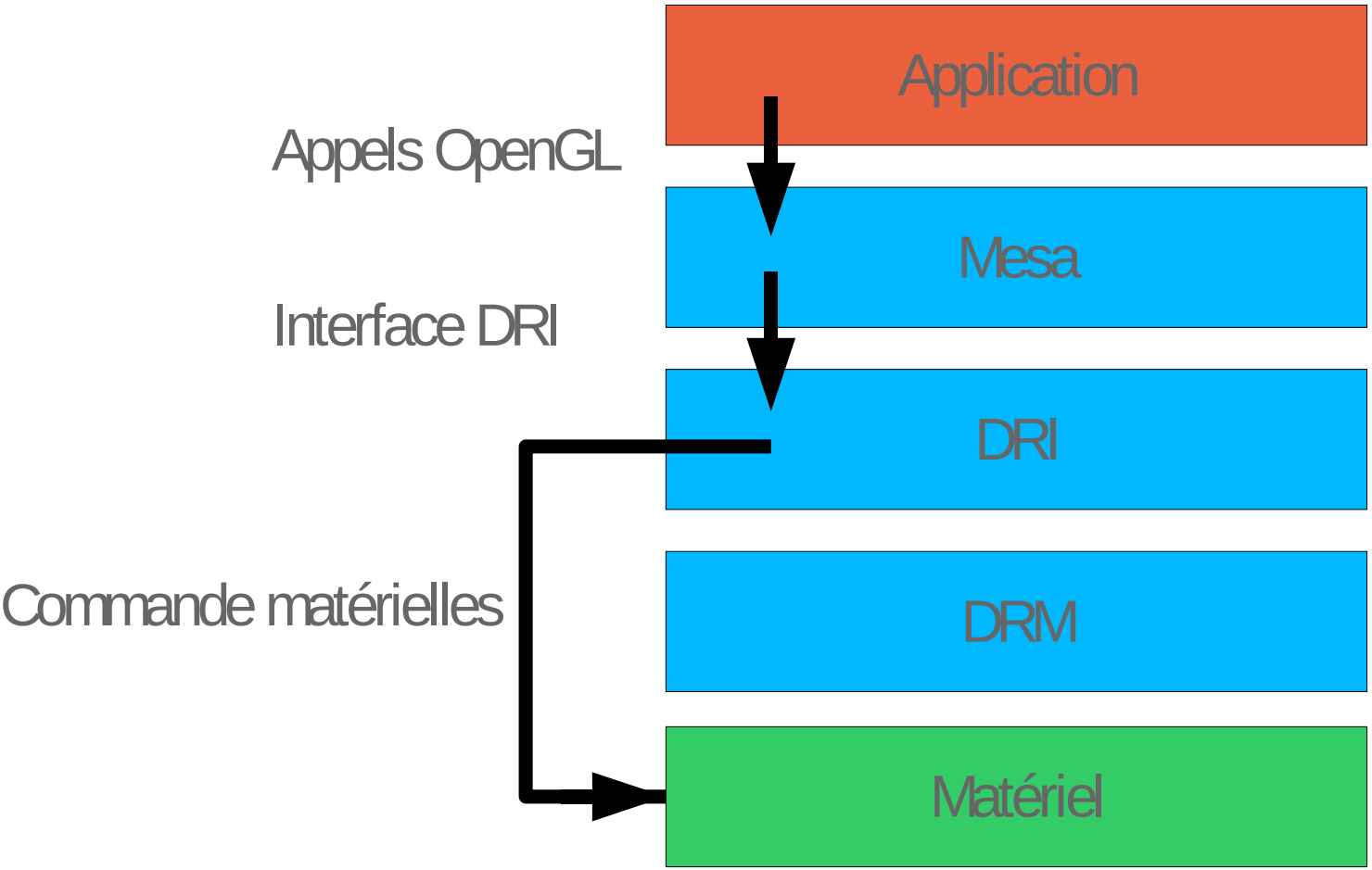


Accélération 3D : Le modèle DRI/DRM

- Fonctionnement du DRI/DRM
  - Inconvénient : beaucoup de couches à traverser
  - En particulier, le passage dans le noyau est très coûteux

Accélération 3D : Le modèle DRI/DRM

- Fonctionnement du DRI/DRM dans NouVeau

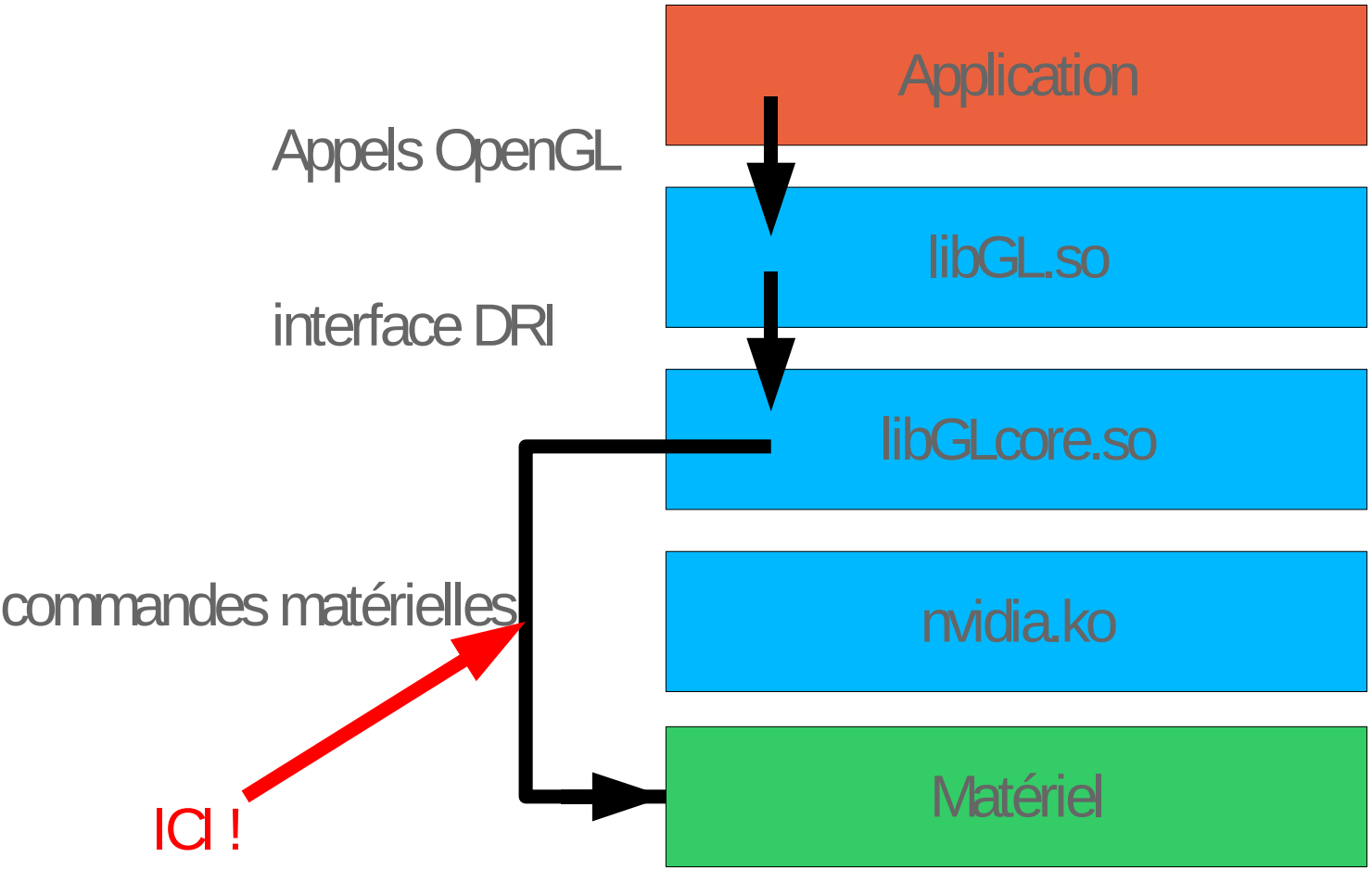


Accélération 3D : Le modèle DRI/DRM

- Quid de la sécurité ?
  - Le DRM vérifie les paquets envoyés à la carte
    - ➔ Enlève les commandes qui pourraient poser des problèmes de sécurité
  - Dans le cas des cartes Nvidia, le matériel lui-même peut vérifier les opérations effectuées pas le DRI

Ingénierie inverse

- Nécessité de comprendre les fonctionnalités non documentées



Ingénierie inverse

- Solution
  - Trouver par où passent les commandes envoyées au matériel
    - ➔ Trouver la FIFO
  - Espionner les changements dans cette FIFO
    - ➔ Regarder le contenu de la FIFO
    - ➔ Provoquer un changement
    - ➔ Comparer le nouvel état de la FIFO avec l'état précédent
  - Espionner les changements dans les registres

Ingénierie inverse

- Concrètement
  - Créer un processus utilisant OpenGL
  - Trouver la FIFO dans son espace d'adressage
  - Copier le contenu de la FIFO
  - Appeler une ou plusieurs commandes OpenGL
    - glClear()
    - glVertex()
    - ...
  - Comparer la FIFO avec son état précédent
  - Déduire le fonctionnement de l'opération
    - A la main !

Ingénierie inverse

- Exemple
  - Comprendre l'envoi de triangles
    - Envoyer 1 triangle
    - Envoyer 2 triangles
    - ...
    - Envoyer X triangles
    - Envoyer des triangles dont les sommets valent tous 1
    - Envoyer des triangles dont les sommets valent tous 2
    - ...
    - Envoyer des triangles dont les sommets valent tous X
    - Comparer les résultats et croiser les informations
    - En déduire comment fonctionne l'envoi de triangles

Ingénierie inverse

- Avantages
  - Simple à mettre en oeuvre
  - Ne requiert pas les droits root sur la machine
- Inconvénients
  - On ne peut pas voir tous les changements !
  - En particulier, les écritures de registres faites dans le noyau restent impossibles à suivre

Travaux effectués

- Nettoyage du code du DDX
- Support de EXA
- Ecriture d'un DRM
  - Allocation de mémoire video
  - Gestion des objets
- Ingéniérie inverse des cartes
  - Les fonctionnalités 3D des cartes TNT jusqu'aux Geforce 7x00 sont connues

Travaux en cours

- Les cartes Nvidia ont plusieurs contextes
  - Mais il faut savoir passer d'un contexte à l'autre
    - ➔ Nécessaire pour pouvoir avoir plus d'une application à la fois !
    - ➔ Fonctionne... de temps en temps (circonstances précises à déterminer)
- Le pilote DRI compile mais est incomplet
  - Ecriture d'un pilote basique supportant au moins des triangles sur toutes les cartes
    - ➔ Base d'expérimentation

Travaux futurs

- Dual head
  - Utilisation de la nouvelle API randr 1.2
    - ➔ Permet (enfin) d'ajouter/enlever dynamiquement des écrans sous Xorg
    - ➔ Plus propre que mergedfb
- Implémentation de Xv
  - Amélioration des performances
    - ➔ Pour l'instant les données video sont copiées avec une boucle for()

Conclusions

- L'ingéniérie inverse fonctionne
  - Les fonctionnalités 3D sont comprises
- L'ingéniérie inverse fonctionne presque toujours
  - Les opérations effectuées dans le noyau sont impossible à tracer
- L'équipe de développement a beaucoup grandi depuis mars (1 dev permanent -> 5 devs permanents)
- Utilisation de modèles de programmation audacieux
  - Le pilote 2D dépend du DRM
  - Le modèle DRI/DRM a été modifié pour prendre en compte les spécificités matérielles

Conclusions

- “It's so hard to write a graphics driver that open-sourcing it would not help”

  Andrew Fear, Nvidia software product manager

  http://lwn.net/Articles/180633/
- On s'en sort bien, merci !

Questions ?

http://nouveau.freedesktop.org