# DkStreaming : Moto Racer Trafic

**Ce document présente l'étude technique d'un algorithme, permettant une gestion simple et efficace du streaming des niveaux de jeu de Moto Racer Trafic. Cet algorithme doit être mis en œuvre rapidement (de l'ordre d'une semaine) et permettre le streaming d'un décor d'une qualité graphique au moins égale à celle de Burnout 2, malgré les contraintes de mémoire et de vitesse de média imposées par la PlayStation 2.**

## *Sommaire*



Frantz RAIA
version 0.5 (2003/10/23)

## Historique de ce document

| Version | Date | Commentaires |
|---|---|---|
| 0.5 | 2003/10/23 | • Corrections de l'exemple du format de données.<br>• Restrictions à propos de la distance de clipping.<br>• Rappels sur l'organisation optimale des données. |
| 0.4 | 2003/10/22 | • Eléments de réflexion additionnels dans l'analyse. |
| 0.3 | 2003/10/21 | • Première version publique. |

# 1 Besoins

Cette section définit les besoins du projet à réaliser par le secteur programmation.

Un peu de vocabulaire avant de commencer :

| | |
|---|---|
| ***niveau*** | Un niveau complet de jeu |
| ***secteur*** | Les niveaux sont découpés en secteurs |
| ***streaming*** | Module chargeant / déchargeant dynamiquement des secteurs |
| ***média*** | Support physique du jeu (CD / DVD / Disque dur / Réseau) |
| ***seek*** | Déplacement physique de la tête de lecture sur le média |
| ***observateur*** | Point d'observation courant dans le jeu (ex: caméra qui suit le joueur) |
| ***frame*** | Correspond à une itération de la boucle principale du jeu |

Remarque :

Il est important de bien comprendre que le streaming ne (dé)charge pas forcément les données en fonction de la position du joueur, mais en fonction de la position de l'***observateur*** (qui n'est pas forcément la même).

## 1.1 Définition de l'équipe d'étude

Le tableau suivant représente l'équipe d'étude chargée du travail décrit par le présent document :

| **Clients** | | |
|---|---|---|
| **Moto Racer Trafic en particulier, tout jeu du même type en général** | | |
| **Fournisseurs** | | |
| Frantz Raia | Programmeur | f.raia@dokidenki.com |
| **Suivi** | | |
| Jean-Christophe Capdevila | Lead Programmeur | jc.capdevila@dokidenki.com |
| Patrick Bricout | Technology Lead DSI | p.bricout@dokidenki.com |
| Sylvain Grosdemouge | Lead MRT | s.grosdemouge@dokidenki.com |
| Yannick Turbé | Technology Lead | y.turbe@dokidenki.com |

## 1.2 Quels sont les besoins ?

Moto Racer Traffic (***MRT***) doit offrir au joueur des ***niveaux*** détaillés de taille conséquente, malheureusement trop grands pour tenir d'un seul tenant en mémoire (en particulier celle de la PS2).

Nous devons donc être capables de charger en temps réel des portions de niveaux (***secteurs***) en fonction des déplacements du joueur. Les chargements doivent s'effectuer en arrière plan sans que cela n'affecte en quoi que ce soit le reste du jeu (affichage incomplet, saccades, problème de jouabilité, …).

Le premier algorithme développé dans l'urgence, destiné à gérer les secteurs streamés de MRT, n'est pas très économe vis-à-vis de la mémoire. Nous devons donc - comme prévu - développer un nouvel algorithme plus abouti et permettant d'utiliser au mieux la faible quantité de mémoire notamment disponible sur PS2.

## 1.3 Qu'est-ce que ce développement va apporter ?

Ce développement apportera une meilleure gestion des secteurs streamés, et par conséquent, un gain important de mémoire par rapport à l'algorithme actuellement utilisé.

Ce qui permettra a MRT de fonctionner sans problème sur une PS2 (32 Mo de RAM) malgré une distance de clipping importante (par exemple 500m).

## 1.4 Qu'est-ce que ce développement n'apporte pas ?

L'algorithme développé n'améliorera pas les performances du streaming. De toute façon, ces dernières couvrent déjà les besoins actuels de MRT.

Ce développement n'apporte aucune amélioration en ce qui concerne la gestion des instances. Car les instances sont gérées par une autre partie de DkStreaming, et non pas par l'algorithme de gestion des secteurs.

## 1.5 Qui va utiliser cette application ?

Le nouveau module de gestion des secteurs sera programmé proprement et intégré officiellement à DkStreaming. Il sera principalement utilisé par le projet MRT, mais pourra également être utilisé par tout futur projet imposant les mêmes contraintes (typiques des jeux de course).

Le présent document sert de référence en ce qui concerne le format de description des niveaux de jeu streamés par MRT.

# 2 Cahier des charges

Cette section décrit plus en détail les développements à effectuer afin de répondre aux besoins énoncés.

## 2.1 Objectif

### 2.1.1 Fonctionnalités

L'objectif de ce développement est d'offrir dans les plus brefs délais, un nouvel algorithme de gestion des données streamées dans MRT.

Le nouvel algorithme doit tirer parti de l'architecture particulière des niveaux de MRT (du type ***ride***). Cet algorithme doit permettre une gestion fine des ressources mémoires en offrant notamment la possibilité de préciser (manuellement ou automatiquement) pour chaque secteur, quel sont les secteurs limitrophes à charger en mémoire.

L'algorithme doit supporter jusqu'à 4 observateurs simultanés en vue de l'implémentation d'un mode multi joueurs en écran partagé.

Nous devons également définir le format de données associées à ce nouvel algorithme.

Le nouveau module, ***CDkStreamingSiblingsQuadsMethod***, sera officiellement inclus dans la base *Techno* de Doki Denki.

### 2.1.2 Contraintes

Ce développement doit respecter les cours délais à nouveau imposés : Il est important de disposer d'une première version utilisable réalisée en une semaine. Afin de prouver rapidement l'efficacité des choix effectués ainsi que la validité des outils développés par les codeurs scénariques.

L'algorithme doit respecter notamment la principale contrainte matérielle de la PS2 : La faible quantité de mémoire disponible.

## 2.2 Identification des besoins

### 2.2.1 Environnement matériel

Le développement est destiné à Moto Racer Traffic et doit donc fonctionner principalement sur les plates-formes suivantes :

| Plate-forme | Médias Supportés |
|---|---|
| Sony PlayStation®2 (PS2)<br>PS2 TooL<br>(plate-forme principale) | CD-R/CD-ROM<br>DVD-R/DVD-ROM<br>Disques réseau (Kit uniquement) |
| PC<br>(plate-forme de développement) | Tous |

*Plate-formes et médias supportés par DkStreaming.*

Toutefois, l'algorithme développé étant indépendant de la plateforme, il doit logiquement fonctionner sans aucun problème sur Game Cube et X-Box et toute future plateforme.

## 2.2.2 Environnement logiciel

Le code fourni est développé à l'aide des outils suivants :

- Metrowerks CodeWarrior 3.51 (C++ / Consoles) ;
- Microsoft Visual.Net 2002 (C++ / PC) ;

Ce nouvel algorithme baptisé `CDkStreamingSiblingsQuadsMethod` est intégré au sain des librairies internes de Doki Denki.

Les outils de préparation des données (non fournis par le présent développement) seront probablement développés sous MAX afin d'offrir aux codeurs scénariques et utilisateurs un confort maximal.

Ce point reste à la discrétion des codeurs scénariques.

## 2.2.3 Technologies existantes

### 2.2.3.1 DkStreaming

Le développement de l'algorithme de gestion des secteurs d'MRT repose tout naturellement sur l'architecture actuelle de DkStreaming, module chargé du streaming temps réel des niveaux de jeu.

`CDkStreamingSiblingsQuadsMethod` dérivera donc de l'interface `IDkStreamingMethod`.

### 2.2.3.2 DkXMDRead

Le module DkXMDRead est utilisé pour l'interprétation des données au format XML présentes dans les fichiers de définition des niveaux.

### 2.2.3.3 DkMemory

DkMemory est utilisé indirectement pour les allocations mémoires alignées ou non sur consoles.

#### *2.2.3.4 RenderWare et autres middlewares*

Comme `CDkStreamingSiblingsQuadsMethod` est un algorithme logique indépendant des données streamées, il n'utilise pas RenderWare ni aucun autre middleware.

## *2.2.4 Développements à effectuer*

Le développement se fait sur PC pour des raisons de productivité. L'architecture reste cependant conçue et optimisée tout particulièrement pour la PS2 qui impose les plus grosses contraintes techniques (surtout en ce qui concerne la gestion de la mémoire).

Nous devons :

- Définir l'algorithme de `CDkStreamingSiblingsQuadsMethod` ;
- Définir l'interface avec le client :
  - Initialisation ;
  - Déclaration du niveau ;
  - Mise à jour (à chaque frame) ;
  - Libération des données ;
  - …
- Définir le format XML de déclaration des secteurs composants un niveau de jeu ;
- Implémenter `CDkStreamingSiblingsQuadsMethod` ;
- Effectuer des tests et mesures de performances en condition réelle à l'aide d'un jeu d'essai gracieusement fourni par nos chers codeurs scénariques ;

Le module est développé en portant un soin particulier aux points suivants :

- Fiabilité :
  - Support propre des erreurs dans la déclaration des données ;
  - Support propre de mémoire saturée ;
- Performances :
  - Faible empreinte mémoire ;
  - Fragmentation minimale de la mémoire ;
  - Charge CPU minimale ;
- Transparence :
  - Abstraction totale vis-à-vis des données streamées ;
- Aide à l'optimisation des données :
  - Tout comme le reste de DkStreaming, l'algorithme de gestion des secteurs peut fournir à la demande des statistiques pertinentes effectuées en condition réelle.

## *2.2.5 Ce que cet algorithme fait*

#### *2.2.5.1 Interprétation et analyse de l'architecture du niveau de jeu*

C'est le jeu qui charge le fichier de définition d'un niveau courant, mais c'est `CDkStreamingSiblingsQuadsMethod` qui se charge d'interpréter cette définition au format XML.

#### *2.2.5.2 Gestion de la liste des secteurs à (dé)charger*

`CDkStreamingSiblingsQuadsMethod` gère la liste des secteurs à portée de l'observateur. A chaque requête de DkStreaming, il détermine quels sont les secteurs à charger et quels sont les secteurs à décharger.

### *2.2.6 Ce que CDkStreamingSiblingsQuadsMethod ne fait pas*

#### *2.2.6.1 Gestion des instances*

La gestion des instances n'incombe pas à `CDkStreamingSiblingsQuadsMethod`. La gestion des instances n'est pas non plus le sujet de ce document.

## *2.3 Utilisation*

Reportez-vous au manuel de DkStreaming pour plus de précision sur l'utilisation du streaming en général.

Afin d'utiliser le gestionnaire de secteurs `CDkStreamingSiblingsQuadsMethod`, le jeu client doit inclure le header `CDkStreamingSiblingsQuadsMethod.h` dans son projet.

A l'usage, `CDkStreamingSiblingsQuadsMethod` se distingue par les deux différences notables suivantes :

- Le jeu doit fournir une instance de `CDkStreamingSiblingsQuadsMethod` comme gestionnaire de secteurs en paramètre d'initialisation de DkStreaming.
- De plus, les données de définition du niveau de jeu, doivent évidemment être au format de `CDkStreamingSiblingsQuadsMethod` défini plus loin dans ce document.

# 3 Analyse

Cette section décrit les solutions retenues et l'interface de programmation adoptée pour la mise en place du système de streaming.

## 3.1 Solutions retenues

### 3.1.1 Rappel de l'algorithme actuellement utilisé par MRT

#### 3.1.1.1 Principe

L'algorithme actuellement utilisé est très simple : Chaque secteur est affublé d'une sphère englobante. Lorsque cette dernière se trouve à portée de l'observateur le secteur est chargé, dans le cas contraire, le secteur est déchargé.

#### 3.1.1.2 Avantages

La simplicité de ce principe a permis de développer très rapidement une première version de MRT avec décor streamé.

#### 3.1.1.3 Défauts

Cet algorithme présente notamment les défauts suivants :

- Les sphères englobantes sont en général 3 fois plus grandes que les secteurs auxquelles elles sont associées. Par conséquent il y a surconsommation de mémoire (estimée à environ 20%).
- Les secteurs à portée effective de l'observateur sont systématiquement chargés, alors qu'ils ne sont pas forcément visible ou utile au gameplay : Il y a à nouveau une surconsommation de mémoire.

### 3.1.2 Eléments additionnels de réflexion

#### 3.1.2.1 Bsphere ou pas bsphere ?

En y réfléchissant plus, nous pouvons nous rendre compte que les sphères englobantes ne sont pas gênantes lorsqu'elles tombent sur du vide : Cela n'a pas d'effet, et il n'y a pas de consommation inutile de mémoire.

Mais après avoir jeter un œil aux secteurs streamés sous Max, nous pouvons constater qu'en général, les bspheres débordent beaucoup sur les secteurs adjacents (Sur le suivant et le secteur précédent en général).

Ce qui peu provoquer un « sur chargement » des secteurs en limite d'horizon de l'observateur et donc consommer de la mémoire pour rien.

C'est pour cette qu'il semble que l'utilisation d'une délimitation plus sévère permettrait un petit gain de mémoire (pas faramineux, mais même si c'est 15% c'est toujours cela de gagné).

Notons, cependant, les bspheres seront toujours utilisées afin d'effectuer un pré-clipping rapide dans l'algorithme du streaming.

#### *3.1.2.2 Faut-il forcer l'ordre de chargement des données ?*

L'idée est intéressante, mais pas nouvelle : Nous l'avions déjà testée dans Dragon Hunters, mais nous nous sommes rendu compte que le facteur limitatif, se sont les seeks du média.

Il est donc plus important de charger le plus rapidement possible les données depuis le média, plutôt que de chercher à charger en premier le décor le plus proche de l'observateur. Chercher à fixer une priorité en dur du chargement pourrais sérieusement dégrader les performances. Pour s'en persuader, il suffit de penser a un cas extrême : Lorsque les ressources sont organisées en sens inverse sur le média par rapport à l'ordre de chargement forcé.

Ceci dit, surtout avec MRT (circuits majoritairement 1D), il est tout à fait indiqué d'enregistrer les données dans l'ordre de parcours des circuits. Cela provoquera tout naturellement, le chargement des données dans un ordre très proche de celui qui aurait été forcé à la main tout en conservant un débit de données maximal.

### *3.1.3 Concept général du nouvel algorithme*

#### *3.1.3.1 Principe*

Il s'agit donc d'améliorer la situation en prêtant une attention particulière à la surconsommation de mémoire évoquée ci-dessus.

Nous allons donc écrire un nouvel algorithme palliant aux deux principaux défauts de l'actuel :

1. Chaque secteur est identifié par un quad. Ainsi, il n'y aura plus aucun risque de surestimation de surface et donc de surcharge inutile de mémoire.
2. A chaque secteur correspond une liste des secteurs limitrophes à charger simultanément. Cette liste sera définie automatiquement ou manuellement lors de l'export. Les codeurs scénariques peuvent ainsi optimiser les données en choisissant quels secteurs sont présents en mémoire à chaque instant.

#### *3.1.3.2 Avantages*

Les principes de bases restent simples, par conséquent l'algorithme et les outils correspondant peuvent être développés en un temps limité.

Les deux défauts provoquant une relativement importante et inutile surcharge de mémoire sont corrigés. De plus, les codeurs scénariques ont à présent la possibilité de peaufiner eux même quels secteurs charger à chaque endroit du décor.

#### *3.1.3.3 Défauts*

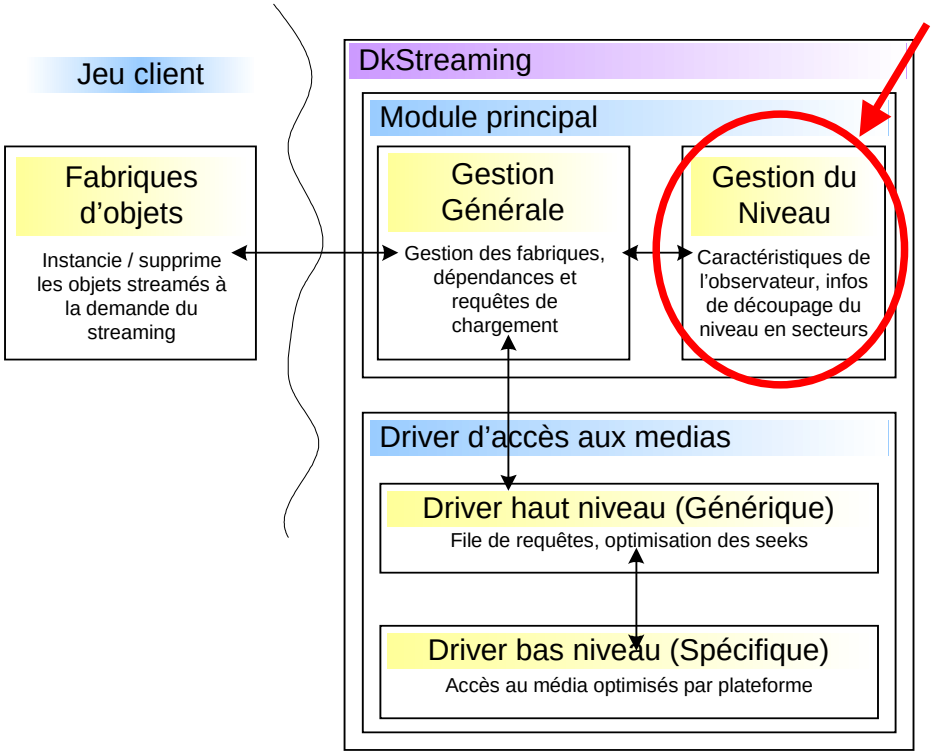
Etant donné qu'il est nécessaire de déclarer pour chaque secteur, la liste de secteurs limitrophes, l'export des données sera plus complexe. De même les réglages seront plus longs.

De plus, il nous sera difficile de proposer au joueur une distance de clipping ajustable (utile sur PC par exemple).

## 3.2 Architecture

### 3.2.1 Architecture générale

Voici où se trouve l'algorithme de gestion des secteurs dans DkStreaming :




En pratique l'intégration de ce nouvel algorithme au sain de DkStreaming et de MRT est très simple :

- Dans DkStreaming : Il suffit de dériver et d'implémenter l'interface `IDkStreamingMethod`.
- Dans MRT : Il suffit de modifier l'initialisation de DkStreaming en précisant que l'on souhaite utiliser la nouvelle méthode `CDkStreamingSiblingsQuadsMethod`.

### 3.2.2 Interface

Note : Seul les membres publics des classes sont présentés ici. Les paramètres des méthodes ne sont pas spécifiés afin de simplifier la lecture du document.

Voici l'interface de `CDkStreamingSiblingsQuadsMethod` :

```
class CDkStreamingSiblingsQuadsMethod:public IDkStreamingMethod
{
public:
                                CDkStreamingSiblingsQuadsMethod();// Default constructor
virtual    inline               ~CDkStreamingSiblingsQuadsMethod(){}// Destructor

virtual    EDkStreamingResult   Init(…);                    // Setup
virtual    EDkStreamingResult   Shut(…);                    // Cleanup

virtual    U32                  GetSectorsCount()const;     // Retrieve sectors count
virtual    CDkStreamingSector * GetSectorsList()const;      // Retrieve sectors list

           EDkStreamingResult   SetObserversPositions(…);// Set current observer position
virtual    EDkStreamingResult   Update(…);                  // Update sectors list

virtual    EDkStreamingResult   GetSectorFromPoint(…)const; // Retrieve sct matching pt
virtual    CDkSector *          GetSectorFromID(…)const;// Sector from ID (loading only)

#if defined(DKSTREAMING_PROFILING_ENABLED)

virtual    EDkStreamingResult   GetProfilingInfo(…)const;   // Retrieve profiling info

#endif // DKSTREAMING_PROFILING_ENABLED

};
```

**CDkStreamingSiblingsQuadsMethod::Shut**

Le streaming appelle cette méthode pour libérer les ressources utilisées par le gestionnaire de découpe.

**CDkStreamingSiblingsQuadsMethod::GetSectorsCount**

Retourne le nombre total de secteurs présent en mémoire.

**CDkStreamingSiblingsQuadsMethod::GetSectorsList**

Retourne un tableau contenant tous les secteurs actuellement en mémoire (voyez le chapitre suivant pour plus de précision à ce sujet).

Cette liste n'est pas dynamique, elle est créée lors de l'initialisation du gestionnaire et détruite à la fin du niveau. Les secteurs sont donc toujours stockés au même endroit en mémoire et sont valables jusqu'au prochain niveau.

**CDkStreamingQuadtreeMethod::SetObserverPosition**

Cette méthode doit être appelée à chaque frame et permet de fixer la position de chaque observateur.

Plus il y a d'observateurs et plus les calculs de secteurs à porté sont longs. L'occupation mémoire maximal augmente également puisque les observateurs peuvent être à des localisations distinctes.

**CDkStreamingSiblingsQuadsMethod::Update**

Met à jour la liste des secteurs en marquant les secteurs à portée et les secteurs hors de portée. Le streaming passe en paramètre toutes les informations utiles concernant les paramètres d'observation.

**CDkStreamingSiblingsQuadsMethod::GetSectorFromPoint**

Permet de savoir à quel secteur appartient un point précis de l'espace. Cette méthode est uniquement destinée au client du streaming est n'est pas utilisée en interne pour la détermination de secteurs à charger.

**CDkStreamingSiblingsQuadsMethod::GetSectorFromID**

Permet de retrouver un secteur grâce à son ID déclaré dans le fichier XML. Attention : Cette méthode est utilisée en interne par DkStreaming et n'est valable et utilisée qu'au moment du chargement pour l'interprétation des données.

#### CDkStreamingSiblingsQuadsMethod:: GetProfilingInfo

Lorsque le streaming est compilé avec la macro `DKSTREAMING_PROFILING_ENABLED` définie, tous ses modules effectuent des mesures de performances qui aident à déterminer les goulots d'étranglements et aident ainsi à optimiser le format et l'organisation des données.

Cette méthode permet d'obtenir en temps réel les résultats des mesures en cours.

Le streaming compilé en mode *profiling* est légèrement plus lent et consomme plus de mémoire. N'oubliez donc pas de désactiver ce mode pour vos *masters* !

## 3.3 Principes de fonctionnement

### 3.3.1 Notes sur l'implémentation

L'implémentation est simple :

A chaque frame, DkStreaming demande à `CDkStreamingSiblingsQuadsMethod` de marquer les secteurs à portée des observateurs.

L'algorithme va chercher dans quels secteurs se trouvent les observateurs. Ces secteurs seront marqués comme étant « à portée ».

Puis, chaque secteur figurant dans les listes associées de secteurs limitrophes, seront également marqués comme « à portée » des observateurs.

Notez donc que plus il y a d'observateurs, et plus il y a de secteurs, plus le rafraîchissement de cet algorithme sera long. Ceci dit étant donné les pré calculs effectués, et le faible nombre de calculs effectués en temps réel, l'occupation CPU devrait rester négligeable.

### 3.3.2 Fiabilité et performances

#### *3.3.2.1 Gestion mémoire*

`CDkStreamingSiblingsQuadsMethod` alloue de la mémoire uniquement lors du chargement de l'architecture du niveau. Autre allocation dynamique n'est effectuée.

De plus, les informations définissants la structure du niveau sont intégralement stockées de manière contiguë dans un seul block mémoire. Cela réduit à la fois la fragmentation, ainsi que les fautes de cache très coûteuses sur PS2.

#### *3.3.2.2 Fiabilité*

Les deux erreurs les plus fréquentes : Mémoire pleine et données invalides, sont correctement gérées lors de l'initialisation du niveau.

De pas la conception simple de l'algorithme, en utilisation courante, aucune autre erreur ne peut arriver.

## 3.4 Format des données

### 3.4.1 Description d'un niveau de jeu (xxSECTORSLST.XMD)

Ce fichier contient deux choses importantes concernant les données streamées :

- Les paramètres qui définissent le format du niveau ;
- La liste des secteurs qui constituent le niveau.

Ce fichier doit respecter la structure suivante :

```
<ROOT>
	<LevelProperties>
		…                                       // Paramètres du niveau
	</LevelProperties>
	<SectorsList>
		<Sector ID="x">                         // Déclaration d'un secteur
			…                               // Paramètres du secteur
			<File FactoryID="x">…</File>    // Un fichier indépendant
			<DependentFiles>                // Liste de fichiers dépendants
				<File FactoryID="x">…</File>  // Un fichier
				<File FactoryID="x">…</File>  // Un autre fichier
				…                       // Autres fichiers
			</DependentFiles>               // Fin des fichiers dépendants
			<DependentFiles>                // Autres dépendants
				<File FactoryID="x">…</File>
				<File FactoryID="x" DataID=”x”>…</File>
				…
			</DependentFiles>
			…                               // Autres fichiers
			<Quad>                          // Quad définissant le secteur
		// 4 points déclarés dans le sens inverse des aiguilles d'une montre
				<Point>
					<X>…</X>
					<Y>…</Y>
				</Point>
				<Point>
					<X>…</X>
					<Y>…</Y>
				</Point>
				<Point>
					<X>…</X>
					<Y>…</Y>
				</Point>
				<Point>
					<X>…</X>
					<Y>…</Y>
				</Point>
			</Quad>
			<Sibling>…</Sibling>            // Secteur limitrophe
			<Sibling>…</Sibling>            // Autre secteur limitrophe
			…                               // Autres secteurs limitrophes
		</Sector>
		…                                       // Autres secteurs
	</SectorsList>
</ROOT>
```

#### Quad

Quad définissant la zone couverte par le secteur (en 2D, vu du dessus). Chaque quad est défini par quatre points, déclarés sans le sens inverse des aiguilles d'une montre.

#### Sibling

Liste des secteurs adjacents. Chaque secteur adjacent est référencé par son ID numérique. Ces secteurs seront obligatoirement chargés lorsque l'observateur se trouve dans le secteur courrant.

Voici un exemple de la description d'un niveau utilisant l'algorithme de découpe de MRT :

```
<LevelProperties>
	…		// Pour l'instant, il n'y a pas de propriétés de niveau
</LevelProperties>
<SectorsList>
	…					// Autres secteurs
	<Sector ID="107">			// Déclaration d'un secteur
		<File FactoryID="0">SDAT010320.xmd</File>
		<File FactoryID="1">SDAT010320.dff</File>
		<File FactoryID="3">SDAT010320.tga</File>
		<DependentFiles>
			<File FactoryID="2" DataID="0">SDAT010320.hbv</File>
			<File FactoryID="2" DataID="1">SDAT010320.hke</File>
		</DependentFiles>
		<Quad>
			<Point>
				<X>0</X>
				<Y>0</Y>
			</Point>
			<Point>
				<X>1</X>
				<Y>0</Y>
			</Point>
			<Point>
				<X>1</X>
				<Y>1</Y>
			</Point>
			<Point>
				<X>0</X>
				<Y>1</Y>
			</Point>
		</Quad>
		<Sibling>95</Sibling>		// Secteur limitrophe
		<Sibling>97</Sibling>		// Secteur limitrophe
		<Sibling>108</Sibling>		// Secteur limitrophe
		<Sibling>109</Sibling>		// Secteur limitrophe
		<Sibling>203</Sibling>		// Secteur limitrophe
		…				// Autres secteurs limitrophes
	</Sector>
	…					// Autres secteurs
</SectorsList>
```

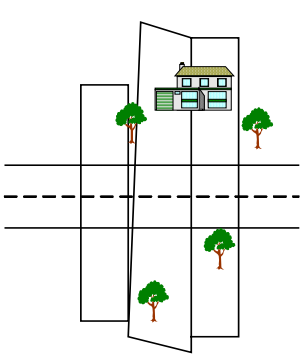
## 3.5 Restrictions

Les restrictions suivantes doivent être respectées afin de garantir le bon fonctionnement du streaming.

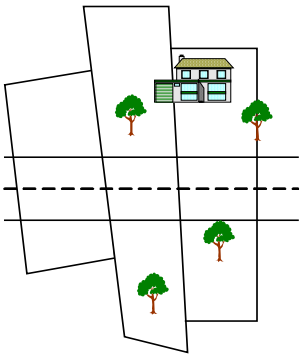
### 3.5.1 Chevauchement

Avec ce genre d'algorithme, il est très important de faire en sorte qu'il n'y ai pas de trou entre les secteurs. Dans le cas contraire, si l'observateur se trouve hors secteur(s), tout le décor sera déchargé !

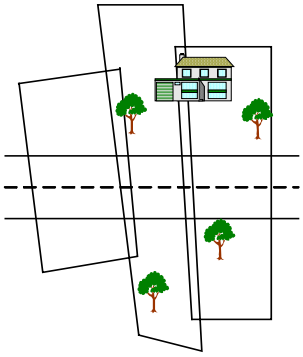
Donc, à la limite, il vaut mieux que les secteurs se chevauchent plutôt qu'il y ai des trous.



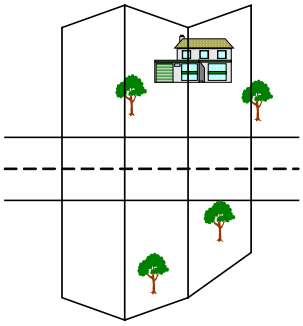
MAUVAIS
Il y a des trous entre les secteurs.



MIEUX
Il n'y a plus de trous. Mais il peut subsister de erreurs mathématiques aux jointures.



BON
Bien que non optimal, le chevauchement permet d'éviter les erreurs mathématiques.



PARFAIT
Les secteurs utilisent les mêmes points, il ne peut donc pas y avoir ni trou, ni erreur mathématique.

## 3.5.2 Observateurs

`CDkStreamingSiblingsQuadsMethod` supporte au maximum 4 observateurs simultanés.

Cette restriction n'est pas très gênante, vu qu'aucun jeu ne permet à plus de 4 joueurs de jouer simultanément sur la même machine.

De toutes façons, au-delà de 4 joueurs, il y a de grandes chances pour que le décor soit intégralement chargé à cause de l'éparpillement !

## 3.5.3 Distance de clipping

### 3.5.3.1 Clipping

Avec cet algorithme, il est nécessaire de mettre à zéro la distance de clipping du streaming. Dans le cas contraire, de nombreux secteurs seront considérés comme étant à porté de l'observateur. Et puisque toutes les dépendances de ces secteurs sont automatiquement chargées, cela risque de provoquer le chargement intégral du niveau de jeu !

#### *3.5.3.2 Hystérésis*

L'hystérésis fonctionne comme d'habitude, et la valeur de 10 mètres par défaut convient tout à fait à Moto Racer Trafic. Il est déconseillé de réduire cette valeur par défaut.

### *3.5.4 Format des données*

Les secteurs doivent être déclarés dans le fichier de définition du niveau dans l'ordre croissant. Il faut commencer par le secteur 0 pour terminer au secteur n-1, sans sauter d'index.

## *3.6 Performances*

### *3.6.1 Note sur les performances*

`CDkStreamingSiblingsQuadsMethod` ne consomme pratiquement pas de CPU. Son principe simple nécessite peu de calculs. De plus, la majorité des données nécessaires au fonctionnement sont pré-calculées lors de l'initialisation du niveau.

### *3.6.2 Rappel sur l'organisation des données*

Afin d'obtenir des performances optimales lors du chargement des données, il est conseillé de classer géographiquement les secteurs par ordre croissant (en commençant par le secteur 0).

Ainsi, lors de la création du PKD, les fichiers pourront être triés par ordre alphabétique (c'est une option DkPKDEdit), cela permettra de réduire à néant les seeks qui auraient été effectués lors du parcours linéaire du circuit.

## *3.7 Evolutions futures*

Aucune évolution n'est prévue pour l'instant.

## *3.8 Conclusion*

Nous venons de voir que grâce à la souplesse de DkStreming, il est très facile de développer de nouveaux algorithmes de gestion des secteurs, adaptés à tout type de jeu.

Le nouvel algorithme apportera un important gain de mémoire, qui devrait même permettre à MRT d'améliorer la qualité de ses graphismes tout en fonctionnant avec les 32 Mo limités d'une PS2.