Niveau de détail avec Irrlicht

par Pierre Schwartz (Mon site)

Date de publication : 31/05/07

Dernière mise à jour :

Cet article va présenter les possibilités de gestion de niveau de détail avec le moteur 3D Irrlicht. Prérequis : C++.

I - Le niveau de détail
I-1 - Le LOD, c'est quoi ?
I-2 - Différents types de LOD
I-2-A - LOD fixe
I-2-B - LOD progressif
II - LOD avec Irrlicht
II-1 - LOD de terrain
II-2 - LOD personnalisé
II-3 - Quid de la mémoire ?
III - Conclusion

Copyright © 2007 - Pierre Schwartz. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

I - Le niveau de détail

I-1 - Le LOD, c'est quoi ?

Comment gérer un objet situé très loin d'une caméra ? Si cet objet possède des milliers de facettes, doit-on toutes les dessiner ? Même si l'objet n'apparaîtra que sur une poignée de pixels ? La gestion du niveau de détail (Level Of Details) répond à cette problématique. L'idée maîtresse d'un niveau de détail est de simplifier un objet en fonction de la distance qui le sépare d'un point de vue donné.

On peut ainsi diminuer le nombre de facettes d'un maillage pour pouvoir l'afficher plus rapidement et décharger la carte graphique. Les performances de l'application 3D n'en seront que meilleures.

I-2 - Différents types de LOD

I-2-A - LOD fixe

Une gestion de niveau de détail par un LOD fixe affichera un maillage ou un autre selon la distance qui le sépare de la caméra. Ainsi, il va falloir charger tous les maillages qui peuvent participer à l'objet et déterminer à chaque frame celui qu'il faudra afficher. La liste des maillages devra être accompagnée d'une liste des distances correspondant à chaque maillage. Chaque maillage sera affichable sur une zone qui lui est propre.

Néanmoins, un oeil averti pourra repérer les changements de forme lors des passages d'un maillage à un autre. Tout l'art du LOD fixe consistera à déterminer les distances de changements de maillages et les maillages eux-mêmes pour minimiser les changements visibles.

I-2-B - LOD progressif

La gestion progressive du niveau de détail peut offrir de meilleurs résultats notamment en termes de visualisation : le maillage se modifie lui-même pour retirer ou rajouter des facettes, toujours en fonction du point de vue de la caméra.

La gestion progressive du niveau de détail est plus complexe à mettre en oeuvre puisqu'il va falloir, entre autres, manipuler directement les facettes et les vertices du maillage. Le premier avantage est que les changements du maillage ne seront presque pas perceptibles pour l'utilisateur. Une telle méthode de LOD nécessitera aussi moins de mémoire puisqu'un seul maillage devra être chargé. Néanmoins, il faudra corriger le maillage à chaque changement de position de la caméra et de l'objet, ce qui peut s'avérer plus coûteux en termes de performances.

Copyright © 2007 - Pierre Schwartz. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

II - LOD avec Irrlicht

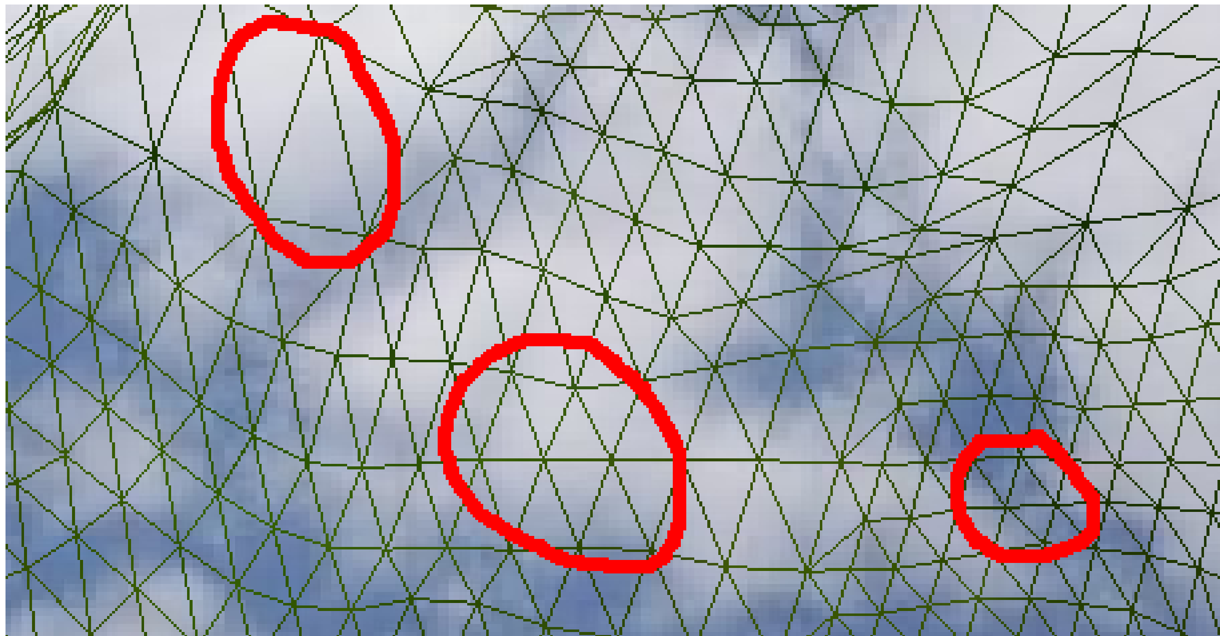
II-1 - LOD de terrain

Irrlicht implémente une gestion de niveau de détail pour l'affichage des terrains. Le pas du maillage est directement réglé par la position de la caméra : il s'agit de la classe ITerrainSceneNode. Voici 2 vues d'un même maillage, en éloignant la caméra. On remarque la diminution du nombre de facettes.

La gestion du niveau de détail est réalisée en prenant un sommet tous les n, n étant dépendant de la distance. Pour pouvoir simplement générer le maillage, Irrlicht essaie de prendre toujours des puissances de 2, il prendra ainsi un sommet sur 2 ou sur 4 ou sur 8 ... selon le niveau de précision souhaité pour afficher le maillage.

On peut presque dire que le maillage est généré 'à la volée'. On peut aussi considérer cette technique comme étant du LOD continu : le maillage se modifie lui-même sans qu'on ait besoin de le pré-calculer. On peut constater le changement de granularité du maillage en faisant un affichage en fil de fer :

Copyright © 2007 - Pierre Schwartz. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.



II-2 - LOD personnalisé

Il faut aussi pouvoir gérer le niveau de détail pour des maillages quelconques, y compris pour des maillages animés. Je vais donc présenter une implémentation d'une telle classe. J'ai choisi de faire du LOD fixe, il faut donc disposer de toutes les versions du maillage à l'exécution. Le principe est très simple :

- On va créer un nouveau type d'objet pouvant s'intégrer dans le graphe de scène
- On va implémenter le choix du maillage à afficher en fonction de la distance qui le sépare de la caméra

Pour ajouter un type personnalisé dans le graphe de scène, il faut le faire hériter d'ISceneNode ou d'une de ses classes filles. J'ai choisi de faire hériter d'IAnimatedMeshSceneNode pour pouvoir gérer les animations. La spécialisation de classes Irrlicht va nécessiter l'implémentation de certaines méthodes notamment les méthodes de pré-rendu et de rendu. Lors du rendu de la scène, Irrlicht va parcourir le graphe de scène et pour chaque objet visible, il appellera sa fonction preRender, puis render et enfin postRender.

Le pré-rendu vérifiera s'il faut changer le maillage courant ou pas, le rendu affichera le maillage sélectionné. La classe est sans surprise : on y retrouve les fonctions virtuelles à implémenter, et les tableaux de maillages et de distances correspondantes. J'ai choisi de stocker les distances à partir desquelles les maillages sont visibles. La première distance doit être 0 ou inférieure.

```
class LOD_Animated_scenenode : public irr::scene::IAnimatedMeshSceneNode{
public:
 LOD_Animated_scenenode(std::vector<irr::scene::IAnimatedMesh*>&,
   std::vector<irr::f32>&,
   irr::scene::ISceneNode *,
   irr::scene::ISceneManager *,
   irr::s32);

 // ISceneNode
 void OnRegisterSceneNode();
 void render();
 const irr::core::aabbox3d<irr::f32>& getBoundingBox() const;
 void OnPreRender();

 // IAnimatedMeshSceneNode
```


Copyright © 2007 - Pierre Schwartz. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

```
    void setCurrentFrame(irr::s32);
    bool setFrameLoop(irr::s32, irr::s32);
    void setAnimationSpeed(irr::s32);
    irr::scene::IShadowVolumeSceneNode* addShadowVolumeSceneNode(irr::s32 id=-1, bool
    zfailmethod=true, irr::f32 infinity=10000.0f);
    irr::scene::ISceneNode* getMS3DJointNode(const irr::c8*);
    irr::scene::ISceneNode* getXJointNode(const irr::c8*);
    irr::scene::ISceneNode* getB3DJointNode(const irr::c8*);
    bool setMD2Animation(irr::scene::EMD2_ANIMATION_TYPE);
    bool setMD2Animation(const irr::c8*);
    irr::s32 getFrameNr();
    void setLoopMode(bool);
    void setAnimationEndCallback(irr::scene::IAnimationEndCallBack* callback=0);
    void setReadOnlyMaterials(bool);
    bool isReadOnlyMaterials();
    irr::scene::IAnimatedMesh* getMesh(void);
    void setMesh(irr::scene::IAnimatedMesh *);
    void setMaterialTexture(irr::s32, irr::video::ITexture*);
    void setMaterialFlag(irr::video::E_MATERIAL_FLAG, bool);

   protected:
    irr::core::aabbox3d<irr::f32> Box;

    int current_index;
    irr::scene::IAnimatedMeshSceneNode* current_node;
    irr::f32 current_distance;

    std::vector<irr::scene::IAnimatedMeshSceneNode*> l_nodes;
    std::vector<irr::f32> l_distances;

    irr::scene::ICameraSceneNode *c;
   };
```

Les maillages doivent être chargés avant l'instanciation des objets LOD_Animated_scenenode. Le chargement se fait de la manière habituelle : par un appel à irr::scene::IAnimatedMesh * irr::scene::ISceneManager::getMesh (const irr::c8 *filename). Il suffira de créer la liste des pointeurs vers les maillages servant pour un objet et de la passer au constructeur de l'objet.

Tous les maillages sont constamment présents dans l'objet, mais un seul a son attribut 'visible' à 'vrai'. Les fonctions intéressantes sont le constructeur, le pré-rendu et le rendu :

Constructeur

```
   LOD_Animated_scenenode::LOD_Animated_scenenode(std::vector<irr::scene::IAnimatedMesh*>& l_m,
            std::vector<irr::f32>& l_d,
            irr::scene::ISceneNode *parent,
            irr::scene::ISceneManager *smgr,
            irr::s32 id)
       :irr::scene::IAnimatedMeshSceneNode(parent, smgr, id){

    // on crée la liste des maillages et des distances
    if (l_m.size() != l_d.size()){
     throw
     return;
    }

    int size = l_m.size();
    l_distances = l_d;

    // on ajoute les éléments au graphe de scène
    for (int i=0; i<size; i++){
     irr::scene::IAnimatedMeshSceneNode* node = smgr->addAnimatedMeshSceneNode(l_m[i], this);
     node->setVisible(false);
     l_nodes.push_back(node);
    }
```


Copyright © 2007 - Pierre Schwartz. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

Constructeur
```
    // seul le premier maillage est visible
    l_nodes[0]->setVisible(true);
    current_index = 0;
    current_node = l_nodes[0];
    current_distance = -1;
    Box = current_node->getBoundingBox();

    setVisible(true);
    smgr->registerNodeForRendering(this);

    c = smgr->getActiveCamera();
}
```

On remarquera que j'ai choisi de stocker le maillage courant, la distance courante ainsi que l'index dans le tableau correspondant à cette même distance et à ce même maillage. Ceci est fait uniquement pour accélérer les accès aux données. Pour la construction, l'utilisateur doit passer les tableaux de maillages et de distances. Par défaut, le premier maillage est activé. On remarquera que les maillages sont chacun créés dans un noeud du graphe de scène. Le graphe de scène contient donc des maillages invisibles.

Le pré-rendu
```
void LOD_Animated_scenenode::OnPreRender(){
 SceneManager->registerNodeForRendering(this);
 irr::core::vector3df &v = getAbsolutePosition()-c->getAbsolutePosition();

 irr::f64 distance = v.getLength();

 if (distance < current_distance){
  // on augmente le détail
  current_node->setVisible(false);
  current_index--;

  current_distance = l_distances[current_index];
  current_node = l_nodes[current_index];
  Box = current_node->getBoundingBox();
  current_node->setVisible(true);
 }else{
  if (current_index+1 < l_distances.size() && distance > l_distances[current_index+1]){
   // on diminue le détail
   current_node->setVisible(false);
   current_index++;

   current_distance = l_distances[current_index];
   current_node = l_nodes[current_index];
   Box = current_node->getBoundingBox();
   current_node->setVisible(true);
  }
 }
}
```

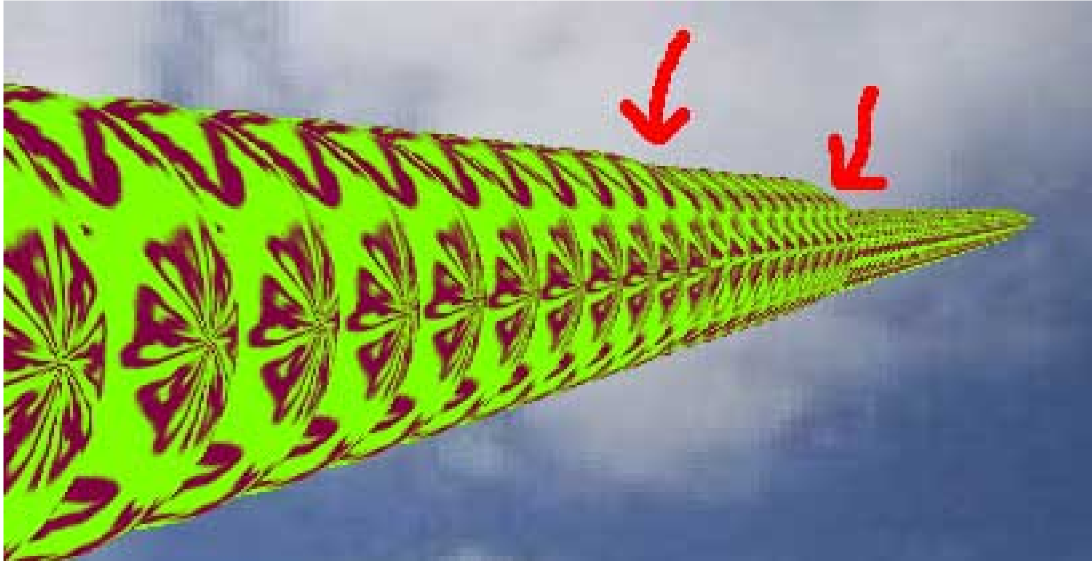
Le pré-rendu se contente de vérifier s'il faut changer le maillage courant ou non. S'il faut le changer, on masque le maillage courant, on détermine le nouveau maillage et on l'active.

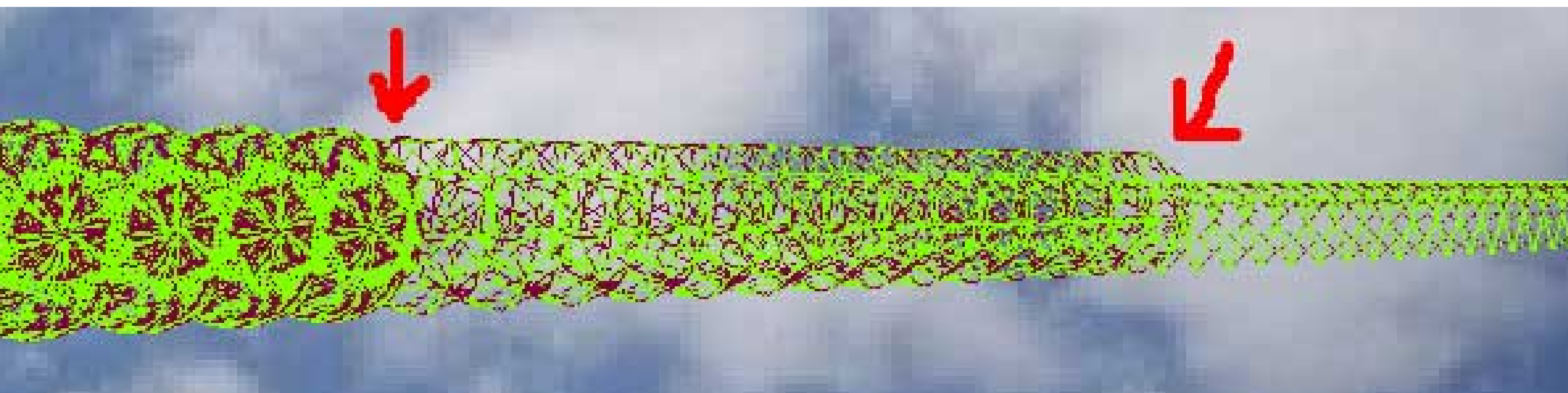
Le rendu est vraiment sans surprise :

Le rendu
```
void LOD_Animated_scenenode::render(){
 current_node->render();
}
```


Copyright © 2007 - Pierre Schwartz. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

J'ai choisi d'aligner plusieurs instances de cette nouvelle classe. On remarque bien sûr que les objets plus éloignés sont moins détaillés. Les changements de maillage sont repérés par les flèches rouges.





Les maillages que j'ai utilisés sont très basiques : il s'agit de trois versions d'une même sphère : avec 5040, 216 et 18 vertices. La version avec 18 vertices est très caricaturale : en effet, bien qu'éloignée, elle souffre d'un trop grand manque de détail, la taille de l'objet n'est plus respectée.

Des mesures de performances nous montrent naturellement que l'affichage avec une gestion de niveau de détail est plus rapide que l'affichage en niveau de détail maximum. Dans mon exemple, la gestion de niveau de détail permet de passer de 504.000 à 1.800 vertices, en multipliant les performances par presque 6.

Toutes les autres fonctions sont relatives aux animations et/ou aux matériaux. Elles se contentent d'appeler les fonctions éponymes de chaque objet fils, de manière à ce que tous les objets soient cohérents entre eux : une animation doit se poursuivre si on change de maillage, de même, une texture doit être appliquée à tous les maillages. Rien d'exceptionnel.

On pourrait essayer de n'ajouter qu'une version d'un maillage à la fois, et de changer ce maillage lors des déplacements de la caméra, les performances purement géométriques sont semblables, mais nous rajoutons un problème pour tout ce qui ne concerne pas la géométrie : l'application des textures, des shaders, des animations doit être refaite à chaque changement de maillage. Les gains de mémoire peuvent donc être perdus par les traitements supplémentaires nécessaires.

Copyright © 2007 - Pierre Schwartz. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

II-3 - Quid de la mémoire ?

- Le LOD de terrain d'Irrlicht a besoin d'avoir deux maillages en mémoire : le maillage originel et le maillage de travail. Il faut toujours conserver le maillage originel pour pouvoir diminuer la granularité du maillage.
- Ma classe de gestion de LOD a besoin de davantage de mémoire que si on ne gérait pas le niveau de détail : chaque objet géré doit être stocké en plusieurs versions. De même, toutes les versions des maillages sont manipulées par le moteur comme s'il s'agissait d'autant d'objets différents. Le gain de performances est réalisé lors du passage de la géométrie à l'API 3D OpenGL/DirectX puisque seuls les objets visibles sont envoyés au pipeline de rendu (une seule version par maillage). Néanmoins, les versions supplémentaires introduites par la gestion de niveau de détail sont allégées en terme de géométrie, elles sont donc beaucoup moins lourdes à manipuler. Un bon design des versions des maillages devrait au moins diviser par 2 la géométrie à chaque version d'un maillage, on arrive ainsi à une utilisation mémoire au maximum doublée.
- Le pré-calcul des différentes versions des maillages au démarrage du jeu n'amènerait aucun gain, autre qu'une occupation disque plus faible. L'utilisation de la mémoire sera exactement la même.

Copyright © 2007 - Pierre Schwartz. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

III - Conclusion

On se rend compte qu'Irrlicht permet de gérer très facilement des niveaux de détails pour des gains de performances intéressants. La spécialisation de la classe ISceneNode est la clef de toute personnalisation d'Irrlicht, qu'il s'agisse de la création d'une classe de gestion de niveau de détail ou de gestion de n'importe quel type d'objet graphique.

Voici les fichiers de la classe de gestion de niveau de détail :

- **LOD_Animated_scenenode.h**
- **LOD_Animated_scenenode.cpp**

Je remercie l'équipe de la rubrique 2D/3D/Jeux pour leurs remarques constructives et **wichtounet** pour la relecture attentive.

Copyright © 2007 - Pierre Schwartz. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

Copyright © 2007 - Pierre Schwartz. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.