Gérer un graphe de scène 3D

—

Programmer avec Inventor

Jean-Yves Didier

Université d'Evry

8 février 2010

Inventor : historique

Historique :

- Origines : IRIX Inventor (SGI – 1989) ;
- Licences de commercialisation tierces :
  - TGS : années 90 (toujours commercialisé) ;
  - Changement de nom : OpenInventor ;
- Code source originel disponible en GPL (2000).

Différentes implémentations existent :

- OpenInventor 2.0 (SGI) ;
- OpenInventor 8 (Mercury (VSG) – TGS) ;
- Coin 3.1.2 (SIM).

Motivations

OpenGL : bibliothèque de très bas niveau :

- Primitives prévues pour être directement cablées !
- Automate pilotant un ensemble de pipelines ;
- Modifier l'état de l'automate $\Longleftrightarrow$ Modifier le rendu ;
- Géométrie simple : points, lignes, polygones.

Inventor : apports et objectifs :

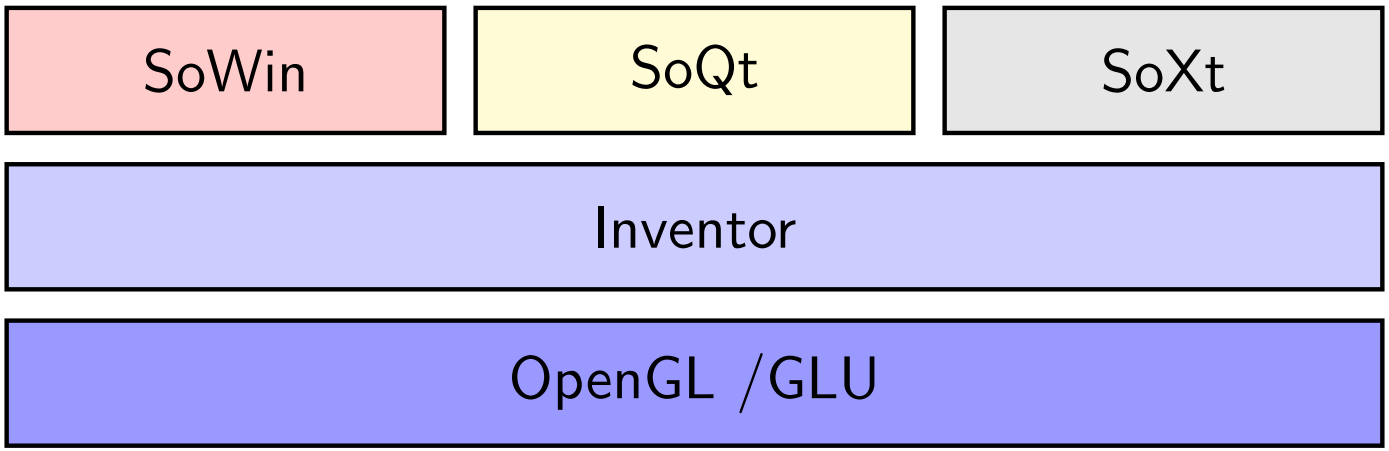
- Notion de **graphe de scène** (plus intuitive) ;
- Bibliothèque orientée objet (C++) ;
- Prend en charge la gestion de l'automate OpenGL ;
- Simplifie l'écriture de programmes 3D.

Utilisation potentielle

- Visualisation rapide de graphes de scène ;
- Manipulation d'entités 3D :
  - Sélection d'objets, modification des propriétés ;
  - Calcul de boîtes englobantes (détection de collision) ;
  - Recherche d'entités.
- Fonctions d'entrée/sortie :
  - Format d'échange natif Inventor (extension .iv) ;
  - Import/export VRML, VRML2 ;
  - Autres suivant implémentation.
- Animation 3D.

Structure

- Indépendant du système de fenêtrage ;
- Lié à d'autres bibliothèque pour l'affichage.



Organisation de la bibliothèque

Plusieurs modules

- Base de données propre à la scène 3D (*scene database*). Contient les éléments utilisés dans le graphe de scène ;
- Bibliothèque de *nodes kits*, des éléments préfabriqués de groupement des noeuds ;
- Bibliothèque de *manipulators*, employés pour interagir avec la scène.

Organisation de la bibliothèque

Plusieurs modules

- Base de données propre à la scène 3D (*scene database*). Contient les éléments utilisés dans le graphe de scène ;
- Bibliothèque de *nodes kits*, des éléments préfabriqués de groupement des noeuds ;
- Bibliothèque de *manipulators*, employés pour interagir avec la scène.

Pour le projet robotique ...

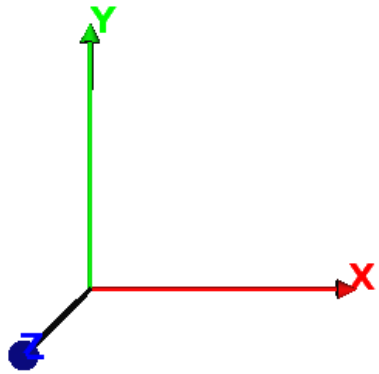
Nous nous intéresserons surtout au graphe de scène !

Conventions utilisées

- Les classes d'objets : `SoXyz`

  `SoMaterial, SoCube, SoRotationXYZ, SoSeparator`
- Les types de base : `SbXyz`

  `SbBool, SbCylinder, SbColor, SbLine, SbMatrix`
- Méthodes : `minMaj`, constantes : `MAJ` ;
- En-têtes : un par objet employé !
- Repères :
  - Les angles sont en radians ;
  - C'est un repère main droite (trièdre direct) ;
  - Voir ci-contre.



Programme type

Structure

```
#include <cheminComplet/SoNomClasse.h>
int main (int argc, char *argv[])
{
   // Initialise la fenêtre de rendu
   // Description de la scène et des comportements
   // Définition de la Caméra
   // Définition de la zone d affichage
   // Affichage à partir du noeud principal
   // Boucle principale
}
```

Exemple simple

Inclusions

```
#include <Inventor/Qt/SoQt.h>
#include <Inventor/Qt/viewers/SoQtExaminerViewer.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoCone.h>
#include <Inventor/nodes/SoSeparator.h>
```

Initialisation

```
int main(int argc, char** argv)
{
  QWidget* myWidget= SoQt::init(argc,argv,argv[0]);
```

Construction du graphe de scène

```
// Il s'agit de la création d'un cône rouge.
SoSeparator* root = new SoSeparator;
root->ref(); // Augmenter le compteur de références
SoMaterial*  rouge = new SoMaterial ;
rouge->diffuseColor.setValue(1.0, 0.0, 0.0);
root->addChild(rouge);
root->addChild(new SoCone);
```

Création d'une zone de rendu

```
SoQtExaminerViewer *myRenderArea = new
    SoQtExaminerViewer(myWidget);
```

Associer la scène et la zone de rendu

```
myRenderArea->setSceneGraph(root);
myRenderArea->show();
```

Affichage de la fenêtre

```
SoQt::show(myWidget);
```

Démarrage de la boucle principale

```
SoQt::mainLoop();
return 0;
}
```

Graphe de scène 3D

Definition

Graphe de scène : Organisation hiérarchique des différents éléments composant une scène 3D.

Natures différentes

- Arbre de construction CSG ;
- Octrees ;
- Graphe de scène VRML.

Graphe de scène Inventor

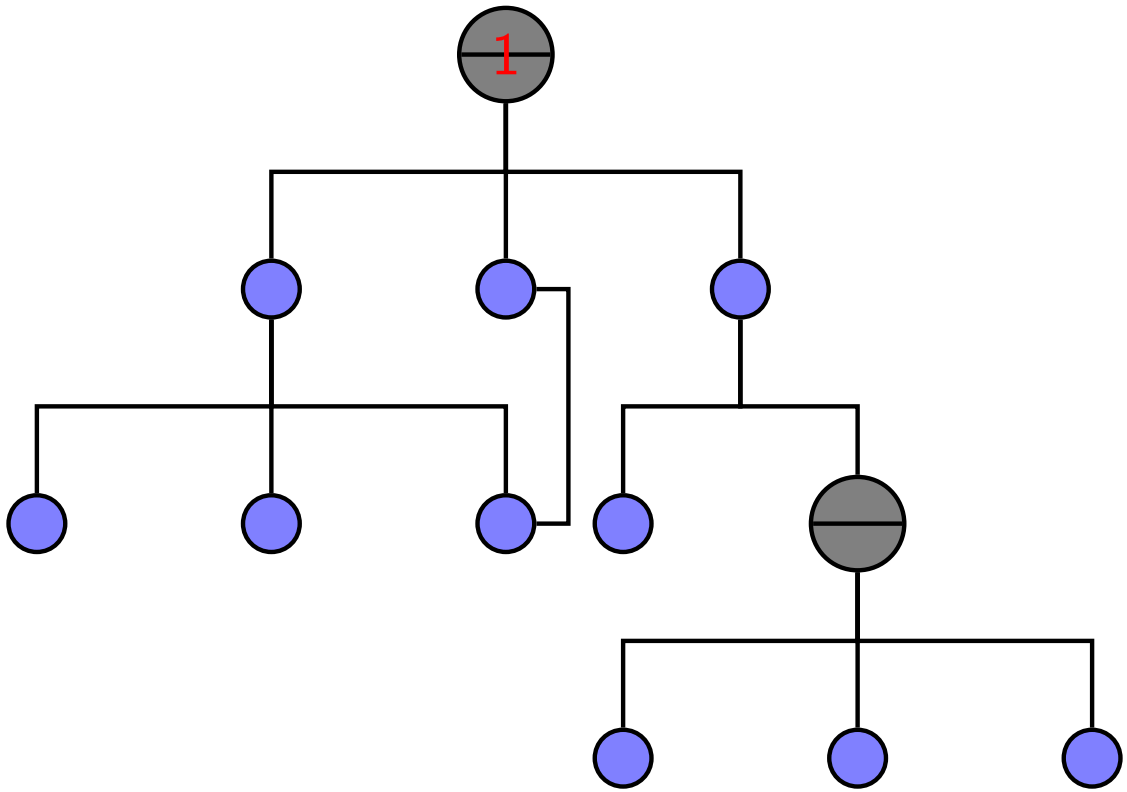
Graphe de scène de type VRML :

- Noeuds, composés de champs ;
- Liaisons noeud à noeud (relation père/fils).

Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

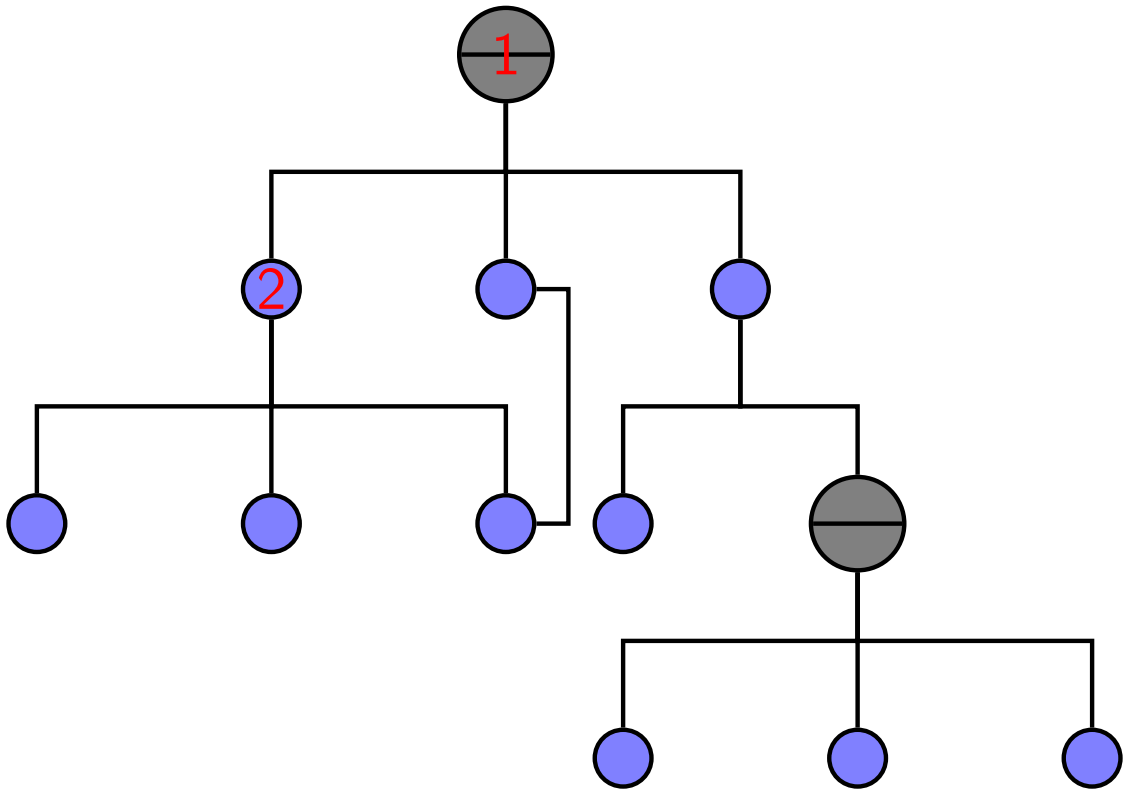
Parcours en profondeur d'abord, en largeur ensuite.



Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

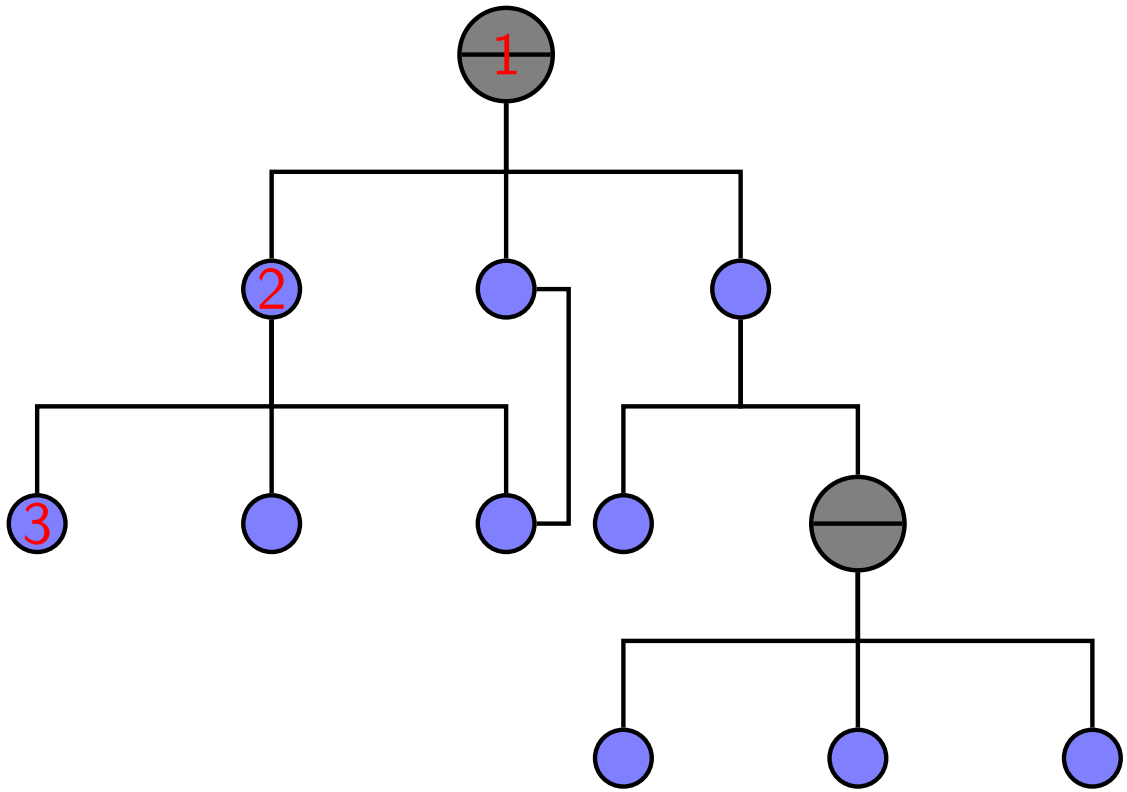
Parcours en profondeur d'abord, en largeur ensuite.



Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

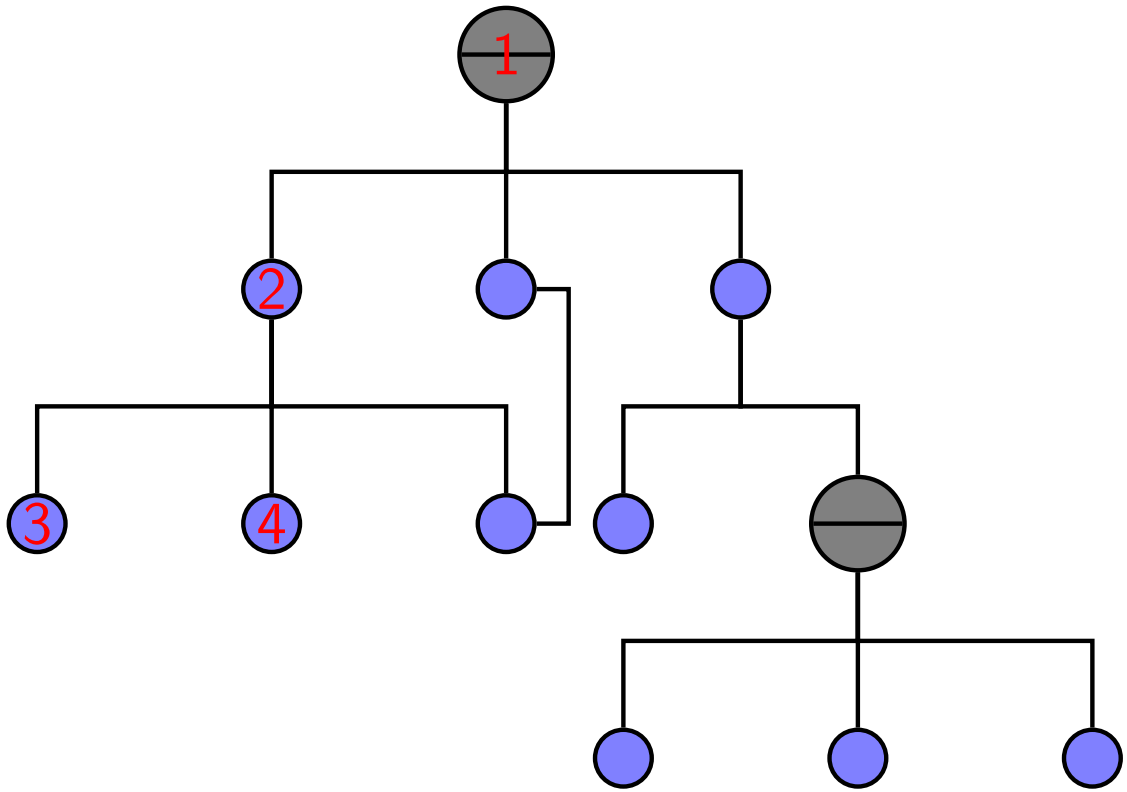
Parcours en profondeur d'abord, en largeur ensuite.



Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

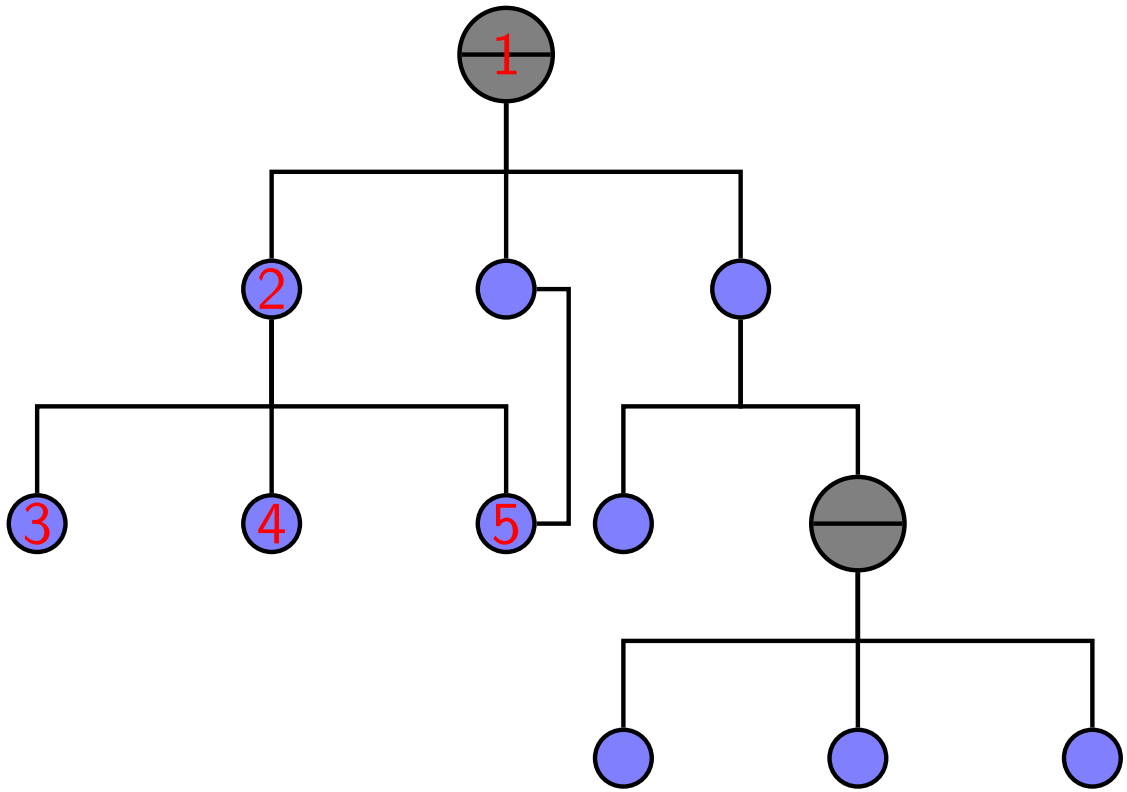
Parcours en profondeur d'abord, en largeur ensuite.



Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

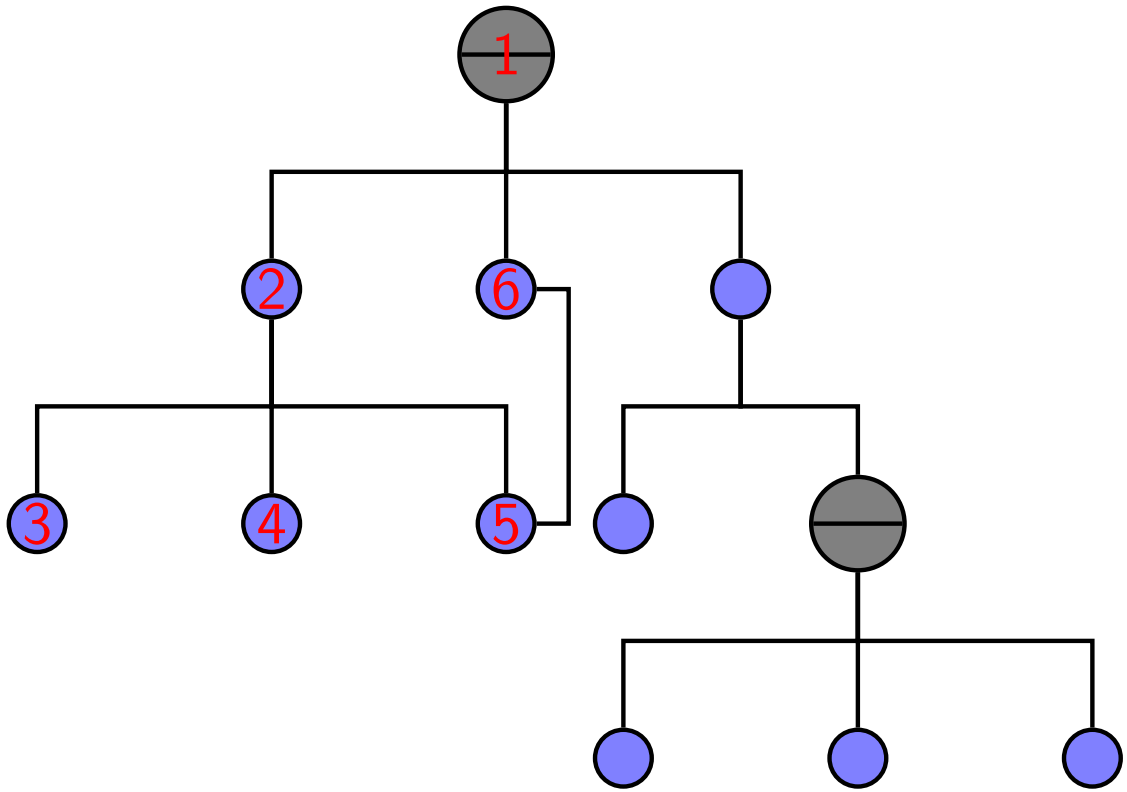
Parcours en profondeur d'abord, en largeur ensuite.



Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

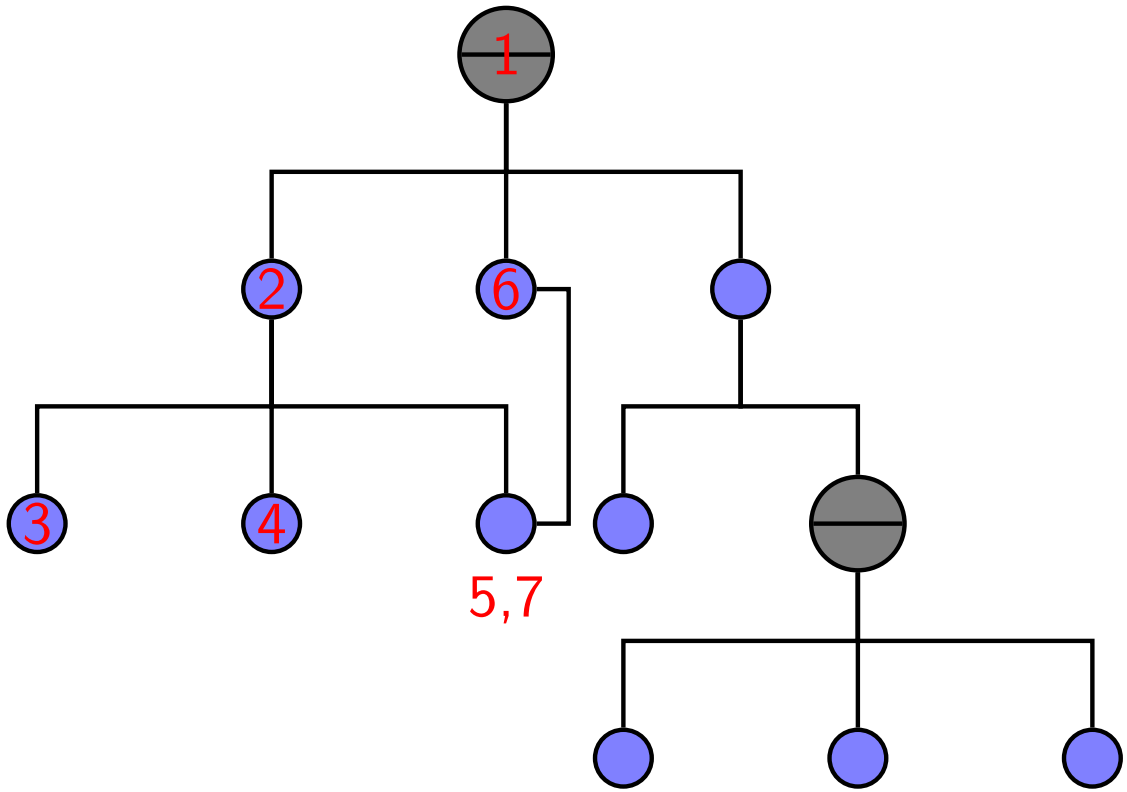
Parcours en profondeur d'abord, en largeur ensuite.



Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

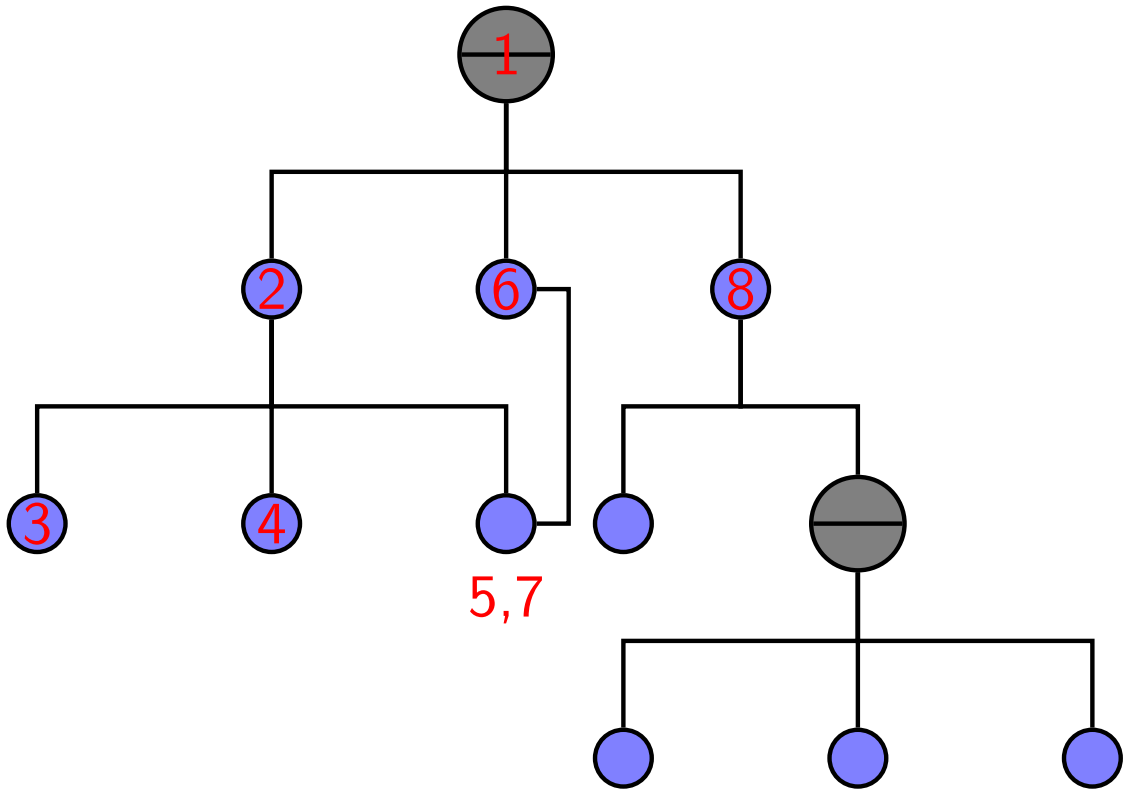
Parcours en profondeur d'abord, en largeur ensuite.



Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

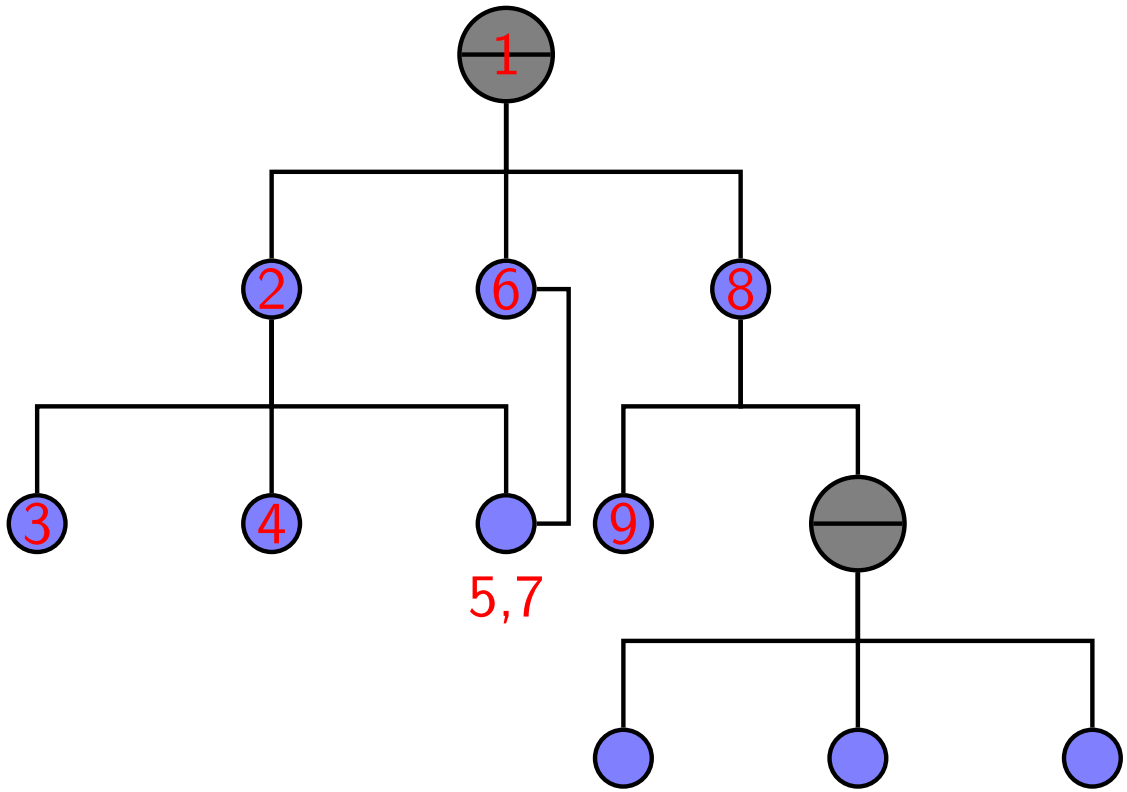
Parcours en profondeur d'abord, en largeur ensuite.



Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

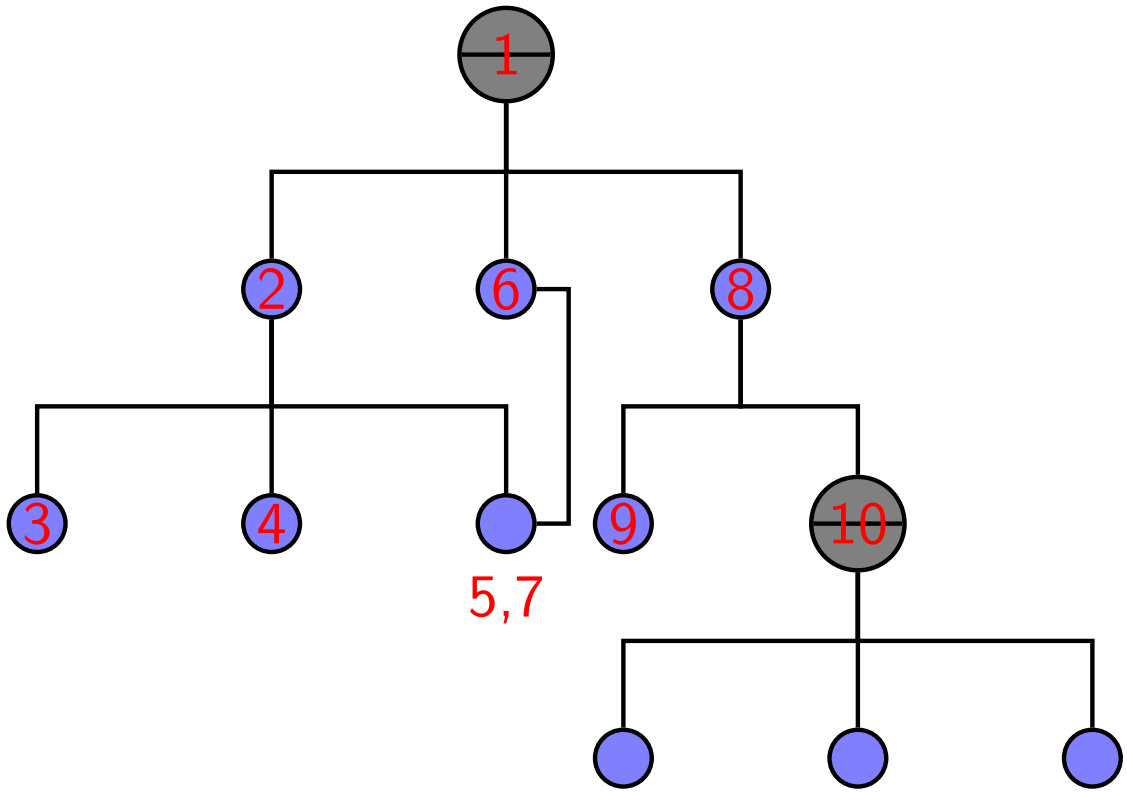
Parcours en profondeur d'abord, en largeur ensuite.



Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

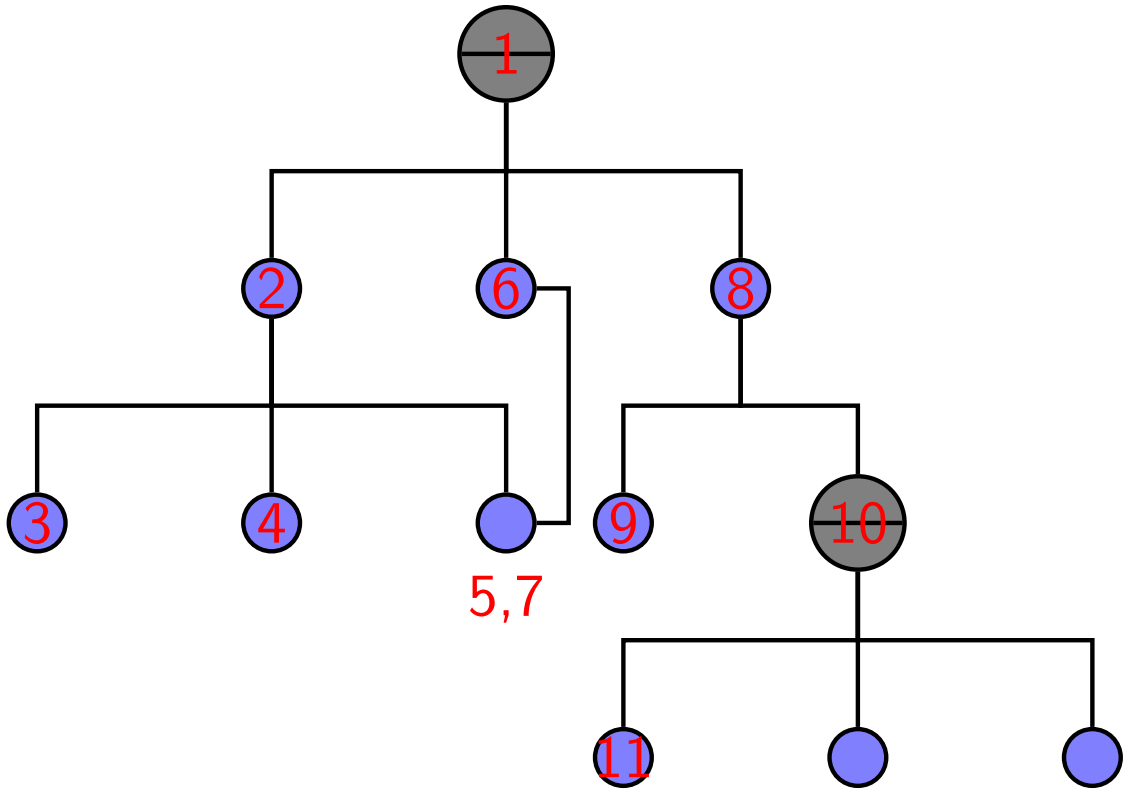
Parcours en profondeur d'abord, en largeur ensuite.



Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

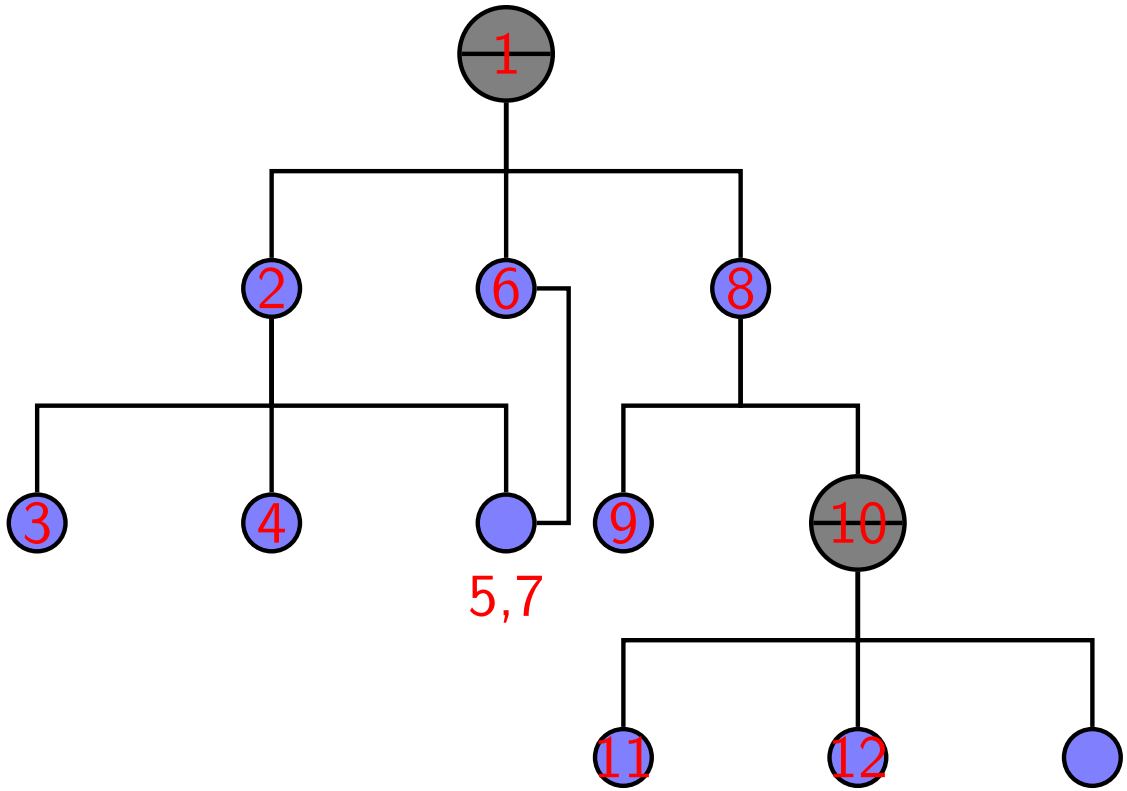
Parcours en profondeur d'abord, en largeur ensuite.



Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

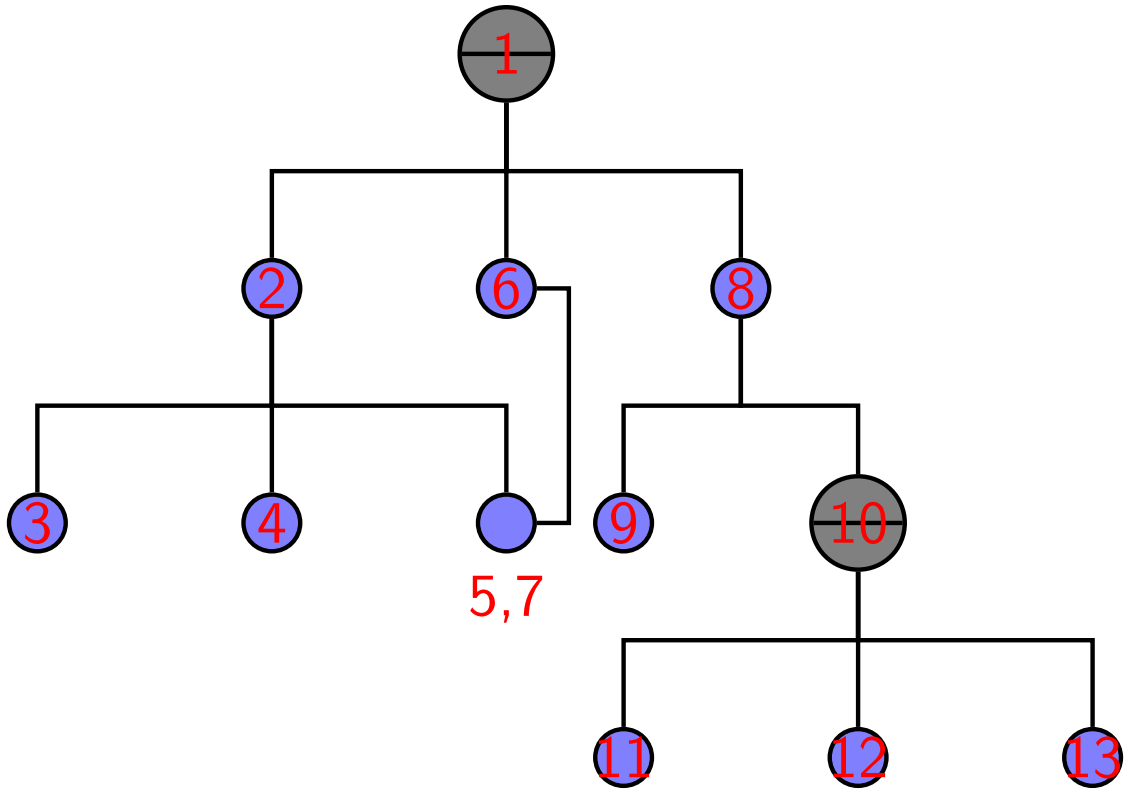
Parcours en profondeur d'abord, en largeur ensuite.



Caractéristiques du graphe de scène

Graphe orienté, acyclique, avec des instance partageables.

Parcours en profondeur d'abord, en largeur ensuite.



Construction d'une scène

Étape 1 : Initialisation

```
SoDB::init();
SoQt::init();
SoInteraction::init();
```

Attention !

Ce sont les premières instructions à utiliser !

Construction d'une scène

Étape 1 : Initialisation

```
SoDB::init();
SoQt::init();
SoInteraction::init();
```

Attention !

Ce sont les premières instructions à utiliser !

Étape 2 : création de la scène

- en chargeant une scène externe ;
- en créant le graphe de scène noeud par noeud.

Les noeuds

3 catégories principales :

- Les formes (*shape nodes*) : contiennent la représentation géométrique des objets ;
- Les propriétés (*property nodes*) : concernent l'apparence des objets et autres caractéristiques qualitatives de la scène ;
- Les groupements (*group nodes*) : contiennent les autres noeuds, sont à la source des embranchements dans le graphe de scène.

Un noeud possède des champs (*fields*) - ce sont ses propriétés.

Création d'un noeud

Instanciation

```
SoSphere* sphere=new SoSphere;
```

Modifier les valeurs d'un champ

Un champ est identifié par un nom et un type.
Les champs ont des valeurs par défaut.
Modifier la valeur : méthode `setValue()` ou `=`

- Types de base : `SbBool, SbVec3f, SbMatrix,` ...
- Énumérations : `SoDrawStyle::LINES`
- Tableaux : `setValues(indice_debut, nb, *tab);`

Champs : exemple

Champs simples SoSFXyz

```
SoTransform* xform = new SoTransform;
SbVec3f vector;
vector.setValue(2.5, 3.5, 0.0);
xform->translation.setValue(vector);
// alternative : xform->translation=vector;
```

Alternative plus directe

```
SoTransform* xform = new SoTransform;
xform->translation.setValue(2.5, 3.5, 0.0);
```

Champs multiples SoMFXyz

```
SoText2* text = new SoText2;
textnode->string.setValue("Toto");
// string=["Toto"];
textnode->string.set1Value(2, "Tutu");
// string=["Toto",...,"Tutu"]
SbString s[2] = { "Titi", "Tata" } ;
textnode->string.setValues(5, 2 ,s);
// string=["Toto",...,"Tutu",...,...,"Titi","Tata"]
// l'opérateur [] est également utilisable.
```

Le compteur de références

Attention !

Ne pas utiliser l'opérateur `delete` pour effacer un noeud !

Le compteur de référence

- Ajout d'un noeud ou d'un sous-graphe : $ref++$;
- Retrait d'un noeud ou d'un sous-graphe : $ref--$;
- Si $ref <= 0$, alors effaçage du noeud ou du sous-graphe ;
- Pour les irréductibles, il y a la manière manuelle :

  `noeud->ref()` : $ref++$ généralement pour les parents ou sous-graphes pour protèger des effaçages intempestifs ;

  `noeud->unref()` : $ref--$ marque potentiellement le noeud pour effaçage.

Noeuds : propriétés génériques

Dérivent de la classe `SoNode`

Peuvent être identifiés par :

- un nom : `setName(nom), getName()` ;
- un type : `isOfType()` .

Construction du graphe, gestion des enfants :

- `addChild(noeud), insertChild(noeud,indice)` ;
- `getChildren()` ;
- (n'a de sens que pour les noeuds de groupement.

Obtenir les champs (introspection) :

`getFields(), getAllFields`

Les noeuds de groupement

Les séparateurs

Cantonnent les effets d'un nom à un groupe
Sauvegarde l'état à l'entrée et le restitue à la sortie :

- transformation géométrique ;
- matériaux (*material*) ;
- modèles d'illumination et lumières ;
- style de dessin ;
- police du texte ;
- coordonnées et normales ;
- caméra.

Utilisation recommandée pour le noeud racine.

Les noeuds de groupement

Les groupes (`SoGroup`) et dérivés

- Sélectionner l'affichage d'un sous-graphe parmi les fils : `SoSwitch` ;
- Gérer un groupe particulier de noeuds : `SoSelection` ;
- Duplication d'un sous-graphe suivant des translations régulièrement espacées en 3 dimensions : `SoArray` ;
- Choisir un sous graphe en fonction du niveau de détail :
  - `SoLOD` : fonction de la distance caméra/objet ;
  - `SoLevelOfDetail` : fonction de la surface affichée.

Camera/illumination

Camera

Noeud de type `SoCamera` :

- Définit le point de vue et le mode de projection ;
- Champs : `position, orientation, aspectRatio, nearDistance, farDistance` ;
- Deux types de caméra :
  - Perspective classique : `SoPerspectiveCamera` ;
  - Perspective cavalière : `SoOrthographicCamera`.

Camera/illumination

Lumière

Noeuds de 3 types dérivés de `SoLight` :

- Directionnelle : `SoDirectionnalLight` ;
- Ponctuelle : `SoPointLight` ;
- Spot : `SoSpotLight` .

Camera/illumination

Lumière

Noeuds de 3 types dérivés de `SoLight` :

- Directionnelle : `SoDirectionnalLight` ;
- Ponctuelle : `SoPointLight` ;
- Spot : `SoSpotLight` .

Remarque

Caméra et lumière sont définis par défaut dans les composants de visualisation dérivés de `SoQtViewer`.

Géométrie

Formes de base :

`SoCube, SoCone, SoCylinder, SoSphere`

A partir de points 3D :

`SoPointSet`

`SoLineSet, SoIndexedLineSet`

`SoTriangle, SoIndexedTriangleSet`

`SoFace, SoIndexedFaceset`

`SoQuadMesh`

Exploitent :

`SoCoordinate3, SoNormal, SoNormalBinding`

Attributs et propriétés

Définir les couleurs selon OpenGL : `SoMaterial` :

`{ambient|diffuse|specular|emissive}Color` ;

`shininess, transparency` .

Style de rendu (`SoDrawStyle`) :

`style` : `INVISIBLE, POINTS, LINES, FILLED` ;

`pointSize` : [0.0, 1.0]

`lineWidth` : [0,255]

`linePattern` : [0, 0xffff]

Modèle d'illumination (`SoLightModel`) :

`model` : {BASE_COLOR | PHONG}

Les transformations

Dérivent de `SoTransform`.
Champs par ordre d'application des calculs :

1. `translation` [1]
2. `rotation` [2]
3. `scaleFactor` [3]
4. `scaleOrientation`
5. `center`

---

1. `SoTranslation`
2. `SoRotation` ou `SoRotationXYZ`
3. `SoScale`

Le texte

Police de texte (`SoFont`) :

`name, size` .

Dans le plan image de la caméra (`SoText2`) :

`string, spacing, justification` .

Sensible aux transformations (`SoText3`) :

`string, spacing, justification` ;
profils linéaires, courbes, ...

Le texte

Police de texte (`SoFont`) :

`name, size` .

Dans le plan image de la caméra (`SoText2`) :

`string, spacing, justification` .

Sensible aux transformations (`SoText3`) :

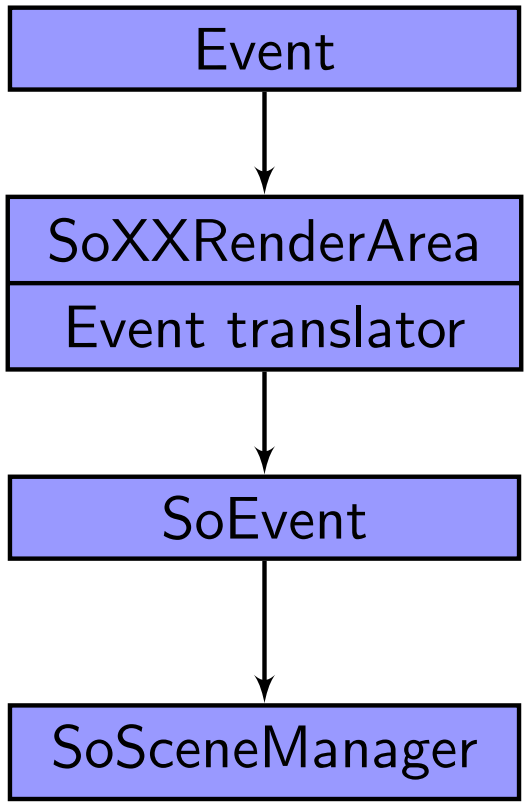
`string, spacing, justification` ;
profils linéaires, courbes, ...

Remarque

Bien d'autres noeud existent mais nous ne les détaillerons pas tous.

Traiter des évènements

De l'OS à Inventor



Les évènements Inventor

Dérivent de `SoEvent` :

`SoKeyboardEvent, SoMouseButtonEvent`, ...

`SoLocation2Event, SoMotion3Event`, ...

Propriété des évènements :

- contient un type : `getType()` ;
- instant d'occurence : `getTime()` ;
- position du curseur : `getPosition()` ;
- état des touches spéciales (Ctrl, Alt, Shift) :

  `was{Shift|Alt|Ctrl}Down()`

Macros pour filtrer les évènements :

`SO_MOUSE_PRESS_EVENT()` ...

Traiter un évènement

4 solutions possibles :

- Utiliser les noeuds d'inventor possédant la capacité de traiter des évènements :
  `SoManipulator, SoSelection,` ...
- Mécanisme d'*event callback* ;
- Mécanisme de *callback* générique à l'aide de `SoCallback` ou à l'aide des capteurs (*sensors*).
- Travailler avec les éléments natifs du système de fenêtrage.

Traiter un évènement

4 solutions possibles :

- Utiliser les noeuds d'inventor possédant la capacité de traiter des évènements :
  `SoManipulator, SoSelection,` ...
- Mécanisme d'*event callback* ;
- Mécanisme de *callback* générique à l'aide de `SoCallback` ou à l'aide des capteurs (*sensors*).
- Travailler avec les éléments natifs du système de fenêtrage.

Attention !

Travailler avec le système d'évènement natif est globalement une mauvaise idée.

Mécanisme d'*event callback*

Création d'un noeud d'*event callback*

```
SoMaterial* mat = new SoMaterial;
SoEventCallback* eventCB = new SoEventCallback;
eventCB->addEventCallback( SoKeyBoardEvent::
    getClassTypeID(), myKeyPressCB, mat) ;
root->addChild(eventCB);
```

Écriture de la fonction de callback

```
void myKeyPressCB(void* userData, SoEventCallback* evt)
{    SoMaterial* mat = (SoMaterial*)userData;
     if (SO_KEY_PRESS_EVENT(evt->getEvent(), UP_ARROW)
     {      mat->diffuseColor(setValue(1.0,0.0,0.0);
            evt->setHandled() ; } ....
```

Les capteurs

Les capteurs dérivent de la classe `SoSensor` :

- ils s'activent :
  - lors de modifications de la scène ;
  - à l'activation d'un *timer*.
- ils invoquent une fonction de *callback* utilisateur.

Côté programme

1. Construire le capteur, sa fonction de *callback* ;
2. Définir la priorité : [0 (fort) . . .1000 (faible)] ;
3. Attacher le *sensor* à un champ ou un noeud ;
4. Supprimer le *sensor* lorsque le traitement est terminé.

Les capteurs

Séquence de fonctionnement interne :

1. Un changement = une notification au capteur (*notify*) lorsque ce dernier est détecté ;
2. Ajout à une queue de traitement (*delay queue*) ;
3. Activation lors du traitement de la queue (*trigger*) ;
4. Suppression de la queue de traitement ;
5. Appel de la fonction de *callback*.

Les capteurs temporisés

L'alarme

Classe `SoAlarm`
Invoque une action de *callback* à un instant donné
`setTime(SbTime)` ou `setTimeFromNow(SbTime)`

L'activation cyclique

Classe `SoTimerSensor`
Invoque une action de *callback* à des intervalles réguliers.
`setInterval(interval)`, `setBaseTime(SbTime)` (par défaut : maintenant).

Les capteurs temporisés

L'alarme

Classe `SoAlarm`
Invoque une action de *callback* à un instant donné
`setTime(SbTime)` ou `setTimeFromNow(SbTime)`

L'activation cyclique

Classe `SoTimerSensor`
Invoque une action de *callback* à des intervalles réguliers.
`setInterval(interval)`, `setBaseTime(SbTime)` (par défaut : maintenant).

Idée d'approfondissement

Voir les moteurs (*engines*).

Action : principe par l'exemple

Le rendu d'une scène

Il est effectué à l'aide d'une action qui :

- est appliquée sur le noeud racine du graphe de scène ;
- parcours le sous-graphe associé à ce noeud ;
- modifie et contrôle l'état de la machine OpenGL suite aux réponses des noeuds ;
- mets à jour la scène et l'affichage.

Action : principe

Une action parcours l'ensemble du graphe de scène pour s'appliquer à l'ensemble de ce dernier, noeud par noeud.

Les actions

Dérivent de `SoAction` et sont de plusieurs types :

- affichage : `SoGLRenderAction` ;
- sélectionner : `SoPickAction` ;
- rechercher : `SoSearchAction` ;
- écrire : `SoWriteAction` ;
- boîte englobante : `SoGetBoundingBoxAction` ;
- fonctions de *callback* : `SoCallbackAction`.

Principe d'utilisation :

- Appel au constructeur ;
- Utilisation de la méthode `apply(...)`.

Exemple partiel d'action : écrire une scène

Construire une scène

```
// scène contenant un cône rouge
SoSeparator* root = new SoSeparator;
SoMaterial* mat = new SoMaterial;
// protection de la scène contre effaçage
root->ref();
root->addChild(new SoDirectionnalLight);
// matériau rouge
mat->setDiffuseColor(1.0,0.0,0.0);
root->addChild(mat);
root->addChild(new SoCone);
```

Exemple partiel d'action : écrire une scène

Application d'une action

```
SoWriteAction action;
action.getOutput()->openFile("scene.iv");
action.getOutput()->setBinary(FALSE);
action.apply(root);
action->getOutput().closeFile();
```

Résultat produit

```
#Inventor v2.1 ascii
Separator {
  DirectionnalLight {}
  Material { diffuseColor 1 0 0 }
  Cone {}
}
```

Le format inventor par l'exemple

Une première scène

Première ligne

```
#Inventor V2.0 ascii
```

Définir un graphe de scène

```
# Toujours commencer par un separator
Separator {
  Cube {
    width 2
    height 1
    depth 2
  }
}
```

Le format inventor par l'exemple

Ajouter une couleur

```
Separator {
  Material {
    ambientColor  0.1 0.06 0
    diffuseColor  0.4 0.25 0
    specularColor  0.75 0.75 0.6
  }
  Cube  { ... }
}
```

Le format inventor par l'exemple

Ajouter une transformation

```
Separator {
  RotationXYZ {
    axis X
    angle 0.707
  }
  Material { ... }
  Cube { ... }
}
```

Le format inventor par l'exemple

Ajouter une lumière

```
Group {
  SpotLight {
    location 2 2 3
    direction -1 -1 -3
    on TRUE
    intensity 1
    color 1 0.8 0.8
  }
  Separator { ... }
}
```

Le format inventor par l'exemple

Animer la scène

```
Group {
  SpotLight { ... }
  Separator {
  RotationXYZ { ...}
  RotationXYZ {
    axis Y
    angle 0 = ElapsedTime
    {
      speed  0.6
    }
    . timeOut
  }
  ...
```

Le format Inventor

Permet de charger des graphes de scène sans écrire les lignes de code équivalentes :

- Chargement de géométrie ;
- Coder seulement les opérations particulières.

L'ancêtre du VRML !

Les implémentations maintenues chargent également les fichiers VRML, VRML2 (97) et X3D :

- `readAll(...)` ou `readAllVRML(...)`
- Des noeuds spécifiques à VRML sont ajoutés (`SoVRMLXyz`)

Le chargement d'un fichier

Charger un fichier

```
SoSeparator* readFile(const char* filename)
{
  SoInput sceneInput;
  if ( !sceneInput.openFile(filename))
  { // Traitement de l'erreur }
  SoSeparator* graph = SoDB::readAll(&sceneInput);
  if ( graph == NULL)
  { // Traitement de l'erreur }
  sceneInput.closeFile();
  return graph;
}
```

Références/Remerciements

Références

- **The Inventor Mentor** – Programming Object Oriented 3D Graphics with OpenInventor
  Josie Wernecke – OpenInventor Architecture Group, Addison Wesley, 1994.
- `http://www.coin3d.org/` : site de Coin3D,
  Implémentation GPL de Inventor maintenue par *Systems in Motion*.

Remerciements

Fawzi Nashashibi (École des Mines de Paris), pour avoir prêté son cours et servi de base à celui-ci.