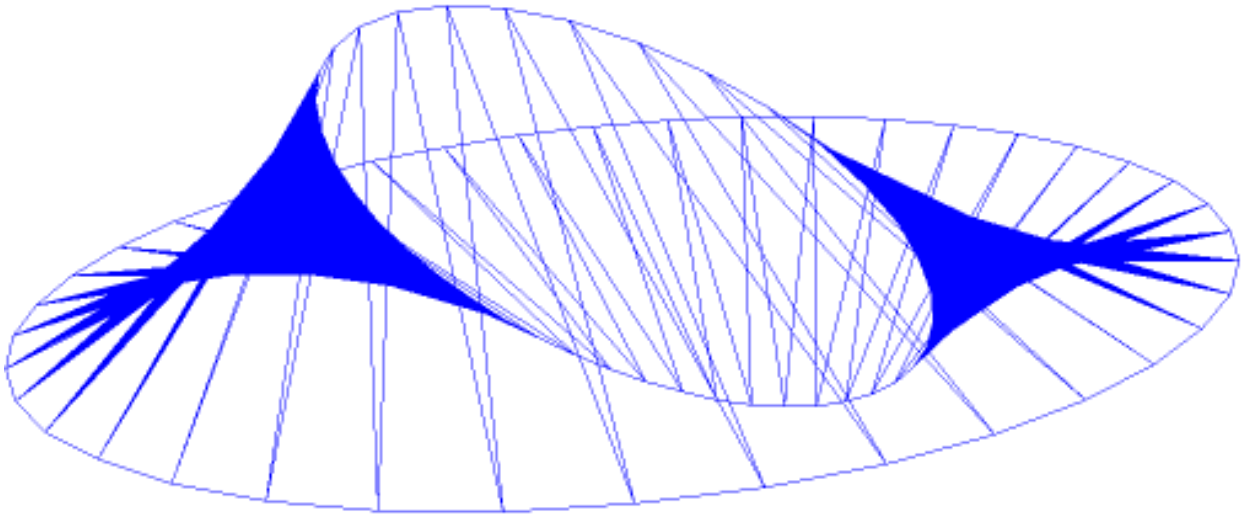
Initiation à l'API Java 3D™

Un tutorial pour les débutants

Chapitre 0
Préface, annexes & lexique



Dennis J Bouvier / K Computing
Traduction Fortun Armel

Le Tutorial de Java 3D v1.5 (API Java 3D v1.1.2)
> traduction fr. version 0.1 <

This documentation was prepared for Sun Microsystems by K Computing (530 Showers Drive, Suite 7-225, Mountain View, CA 94040, 770-982-7881, www.kcomputing.com). For further information about course development or course delivery, please contact either Sun Microsystems or K Computing.

Table des matières

Figures et tableaux

Préface au chapitre 0

Ce document est la préface d'un tutorial sur l'utilisation de l'API Java 3D. Les chapitres supplémentaires de cet ensemble sont disponible à :

Version originale de Sun Microsystems — http://java.sun.com/products/javamedia/3d/collateral

Version française — http://perso.wanadoo.fr/armel.fortun/

Chapitre 0
Préface, annexes & lexique

Bienvenue dans la version 1.0 de la traduction en français du Tutorial de l'Interface pour la Programmation d'Application en Java 3D (l'API Java 3D). Il s'agite de la première version et seulement quatre chapitres sur les huit (Chapitres de 0 à 7) que contient le tutorial de Sun et K Computing ont été traduits et mis en page.

Comme la traduction des chapitres suivants est envisagée, l'historique des mises à jour peut être important pour certains lecteurs. Décrire dans un texte les différences entre chaques versions est assez compliqué, le tableau suivant illustrera l'historique des modifications et des projets de futures versions du tutorial.

version du tutorial	**date**	**nouveaux chapitre(s) traduit(s)**	**modification * des chapitre(s)**
0.1	mars 2000	0,1, 2 et 3	...
0.x	...	chapitres suivants : 4, 5, 6, 7 et 8	...

Tableau 0-1 Historique des modifications et des projets de futures versions du tutorial

* En légende à ce tableau, des modifications sont susceptibles d'êtres apportés aux textes des chapitres. Non pas dans leur contenu (Le tutorial de Sun étant dans sa version définitive)) mais peut-être dans la traduction français/anglais de divers termes ou de corrections orthographiques. Comme vous pouvez le voir dans le tableau 0-1, le tutorial comprend huit chapitres (Chapitres de 0 à 7). Dans les prochains mois, un chapitre supplémentaire sera publié. Pour les mises à jour surveillez les pages web qui se réfèrent à ce fichier ou consultez le groupe de discussion « fr.comp.lang.java ».

0.1 Déroulé du tutorial

Le tutorial se compose d' un ensemble de modules. Chaque module est lui-même un ensemble de chapitres (excepté celui-ci, le Module 0 n'ayant qu'un seul chapitre, le Chapitre 0). Les chapitres d'un module sont liés. Pour des renseignements sur le contenu des modules et des chapitres voir la Partie 0.1.2.

Comme le tutorial est publié par à coup et dans des documents séparés, les rubriques suivantes ont été employées:

- **Annexes :** Afin de rendre la mise à jour plus facile des Annexes en les maintenant, avec le Lexique, au goût du jour, ils sont publiés avec le Chapitre 0 (ce chapitre). Par conséquent, ils apparaissent dans ce document comme des parties numérotées.

- **Numérotation des pages :** Afin de rendre le renvoi à des pages d'un chapitre spécifique, le numéro de la page est précédé par le numéro du chapitre. Par exemple, nous sommes à la page 0-1, qui est la page un du chapitre zéro.

À la publication d'un nouveau chapitre, une nouvelle version du Chapitre 0 (ce chapitre) sera également revue pour ajouter de la matière aux Annexes et au Lexique de ce chapitre.

0.1.1 Contenu du Tutorial

Le tutorial est organisé comme une collection de quatre modules ; Le contenu de chacun d'entre eux détaillé dans les parties suivantes.

Commençant à la page 0-3 la partie nommée «Contenu des chapitres» présente les tables des matières de chaque chapitre.

Tour d'horizon des Modules

Module 0 : Préface, annexes & lexique

Le document que vous lisez actuellement est le Module 0. En plus de la manière de naviguer, il contient une préface, les annexes et le lexique.

Module 1 : Être prêt pour l'API Java 3D

Ce module d'introduction présente les bases de la programmation Java 3D. Le chapitre 1 débute au niveau du programmeur n'ayant jamais utilisé Java 3D. À la fin du chapitre 2, le programmeur novice en Java 3D sera initiéau graphe scénique de Java 3D, à la façon de créer un univers virtuel par la mise en place d'un graphe scénique, à la façon de créer des géométries, les apparences, et programmer des objets visuels pour les utiliser dans des univers vitrtuels simples.

Chapitre 1 : Prendre un bon départ.

Chapitre 2 : Création de la géométrie.

Chapitre 3 : Création facile de volumes.

Module 2 : Interaction et Animation

Le chapitre 4 couvre les bases du comportement, ayant pour but la création de mondes virtuels interactifs. Le chapitre 4 contient l'essentiel sur la navigation dans l'univers virtuel avec la souris, le clavier ainsi que l'interception d'événements. Le chapitre 5 enchaine avec les classes de comportements spéciaux tels que les déterminants, le niveau de détail (LOD), et le billboard afin de créer des objets visuels animés.

Chapitre 4 : Interaction (Traduction à suivre). Existe en Anglais sur le site : http://java.sun.com

Chapitre 5 : Animation (à suivre).

Module 3 : Lumières et Textures

La visualisation de l'univers virtuel est enrichie dans ce module . Par l'utilisation de lumières, des attributs de matériaux, et de textures, le programmeur Java 3D peut créer des objets visuels plus riches sans augmenter la complexité de la géométrie.

Chapitre 6 : Lumières (à suivre).

Chapitre 7 : Textures (à suivre).

Contenu des Chapitres

Voici une table des matières de chacun des chapitres traduits.

Module 1 : Être prêt pour l'API Java 3D

Ce qui n'est pas abordé dans le tutorial

Ce tutorial aborde l'utilisation de l'API Java 3D. Les fonctions utilisées le plus communément par l'API Java 3D sont couvertes. Les propriétés de l'API Java 3D non abordés sont les collisions, les capteurs, la compression de la géométrie, le son spatial et les vues multiples. De nombreux utilitaires de Java 3D distribués avec l'API racine sont abordés, mais pas tous. De plus, les sujets qui ne concernent pas l'API comme les considérations artistiques, les formes d'applications conseillées, et les utilisations non habituelles des visualisations ne sont pas abordées.

0.1.2 Mode d'emploi du tutorial

Les modules sont des collections de chapitres. Toutefois, vous pouvez choisir et regarder les chapitres qui conviennent à vos besoins. En général, les chapitres d'un même module dépendent des chapitres précédents du même module. Par exemple, le Chapitre 2 dépend des connaissances abordées dans le Chapitre 1. De façon analogue, le lecteur du Chapitre 5 est considéré comme étant familiarisé avec le contenu du Chapitre 4. L'enchaînement des modules est représenté dans la figure ci-dessous. Si vous n'avez pas d'expérience avec Java 3D, commencez avec le Module 1 et puis continuez à votre convenance vers le Module 2 ou le Module 3.

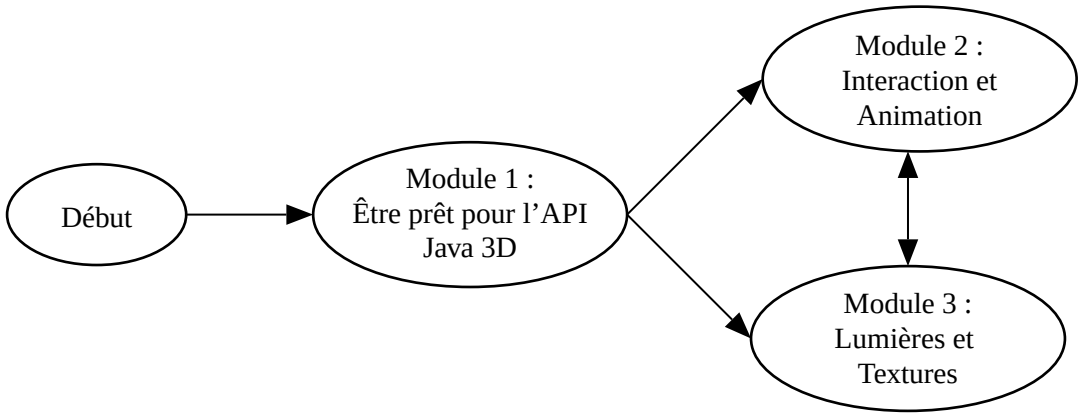


Figure 0-1 Circuit à travers le tutorial de Java 3D

On trouve dans tout le tutorial des *Blocs de référence* — résumés de certaines classes de l'API. Les blocs de références sont fournis dans ce tutorial pour faciliter la lecture, pas pour remplacer le Guide des Spécifications de l'API Java 3D ou toutes autres références. L'exactitude des blocs de référence est vérifiée avant la publication du document. Si vous rencontrez un problème avec un programme, vérifiez bien que vous soyez en possession d'une version récente des Spécifications de l'API Java 3D. Référez-vous aussi à la Partie 2.2 (page 2-4) pour plus d'information sur les Blocs de références.

0.1.3 Préface au Tutorial

Quel est le contenu ?

C'est un tutorial pour la version 1.1.2 de l'API Java 3D. Il est composé de textes (ce document), de plusieurs autres documents de textes et un certain nombre d'applications d'exemple. Le texte du tutorial est disponible dans un fichier au format Acrobat (PDF). Les fichiers PDF contiennent de petites images, des liens, et des signets les rendant facile d'utilisation en ligne. Le fichier peut aussi être lu sur une copie

papier. Toutefois, quelques images sont en couleurs et cette particularité est perdue lors d'une impression en noir et blanc.

Comment télécharger ce document ?

Les fichiers originaux du tutorial sont disponibles en ligne avec les codes sources des programmes d'exemples, tous cela peut être téléchargé depuis l'adresse suivante :
`http://sun.com/desktop/java3d/collateral/`

Quel public ?

Ce tutorial est conçu pour le programmeur Java ayant une petite expérience graphique, avec aucune ou une petite connaissance de Java 3D. Si en plus d'être familier avec Java vous connaissez les termes de pixel, fenêtre de rendu, RGB, et rendu vous avez alors de bonnes bases. Vous n'êtes pas obligé de connaître quoi que ce soi sur le Z-buffer, les transformations 3D ou n'importe quelle autre interface de programmation d'application graphique 3D pour comprendre ce tutorial, mais cela pouvant aider. Dans tous les cas, ce tutorial est écrit pour être accessible à tous.

Retour d'information

Comme avec tous nos produits, nous nous efforçons d'avoir une excellente qualité. Si vous avez n'importe quelle question, commentaire ou une erreur a rapporter, consultez s'il vous plaît la page d'accueil de Java 3D, `http://www.sun.com/desktop/java3d`, pour les informations de contact.

Pour la version française, vous pouvez laisser un message à:

`armel.fortun@wanadoo.fr` avec comme objet : **Tutorial Java 3D.**

Conventions Typographiques

- La `courier` représente les lignes de code, les noms des fichiers et de répertoires de l'ordinateur.
- Les caractères en *italique* sont des mises en relief.
- Les caractères **gras** indiquent des éléments du programme.

Les fonds gris indiquent des Blocs de Références.

Les parties avec un contour double sont des sections avancées.

Les parties avec un contour simple expliquent la nature même de l'information de la documentation.

Logiciel nécessaire

Consultez la page d'accueil Java 3D pour les informations les plus récentes.

Image de couverture

L'image de couverture est un ruban tordu rendu par Java 3D. Le programme est abordé dans la Partie 2-6. Le code est fourni dans les exemples distribués avec ce tutorial.

Abréviations utilisées

e.g. : *exempli gratia* (latin), par exemple.

i.e. : *id est* (latin), c'est à dire.

0.1.4 Déclamations

LES INFORMATIONS FOURNIES DANS CE DOCUMENT, AINSI QUE TOUT LOGICIEL ACCOMPAGNANT CE DOCUMENT LE CAS ÉCHEANT (désignés collectivement par le terme «Note d'application») SONT FOURNIS «EN L'ÉTAT» SANS GARANTIE D'AUCUNE SORTE, EXPRESSE OU IMPLICITE, Y COMPRIS, DE MANIÈRE NON LIMITATIVE, SANS AUCUNE GARANTIE DE QUALITÉ MARCHANDE OU D'ADÉQUATION À UN USAGE PARTICULIER. DANS AUCUN CAS SUN ET SES CONCESSEURS NE SERONT RESPONSABLES DES DOMMAGES SUBIS PAR LE LICENCIÉ PAR LE RÉSULTAT D'UNE UTILISATION, D'UNE MODIFICATION OU D'UNE DISTRIBUTION DU LOGICIEL OU DE SES DÉRIVÉS. DANS AUCUN CAS SUN OU SES LICENCIÉS NE POURRA ÊTRE TENU RESPONSABLE POUR N'IMPORTE QUEL REVENU, BÉNÉFICE OU DONNÉES PERDUES OU POUR DES DOMMAGES DIRECTS, INDIRECTS, SPÉCIAUX, CONSÉCUTIFS, FORTUITS OU PUNITIFS, POURTANT CAUSÉS ET SANS SE SOUCIER DE LA THÉORIE DE RESPONSABILITÉ, PAR L'UTILISATION OU DE L'INCAPACITÉ D'UTILISER LE LOGICIEL, MÊME SI SUN A ÉTÉ INFORMÉ DE LA POSSIBILITÉ DE TELS DOMMAGES.

Vous reconnaissez le fait que le logiciel inclus avec ce tutorial n'a été ni conçu ni approuvé pour être utilisé dans le cadre du contrôle aérien, du transport aérien, de la navigation aérienne et des communications aériennes, ni de la conception, de la construction, de l'exploitation ou de l'entretien d'une centrale nucléaire. Le concessionnaire représente et garantit qu'il n'utilisera pas ou redistribuera le logiciel pour de tels buts.

0.2 (Annexe A.) Sommaire des programmes d'exemple

Ce tutorial est fournit avec un jeu de programmes d'exemple. Chacun des programmes fournit est décrit ci-dessous. Si vous n'êtes pas en possession de ces programmes réfèrez vous à la préface pour les instructions de téléchargement.

0.2.1 HelloJava3D

HelloJava3D est une série de programmes utilisés dans le premier chapitre de ce tutorial. Ces exemples se complexifient au fûr et à mesure et se construisent pas à pas.

examples/HelloJava3D/HelloJava3Da

Ce programme affiche un seul cube coloré et immobile, qui n'est ni transformé ou tourné depuis son origine. Il apparaît alors comme un rectangle simple. Ce programme prétend seulement démontrer les bases de la construction d'un programme Java 3D. Il est aussi utilisé comme une base pour les programmes ultérieurs.

examples/HelloJava3D/HelloJava3Db

Ce programme affiche un seul cube coloré et immobile, mais qui a subit une rotation depuis sa position originale. Ainsi, plus d'une face du cube est visible au rendu. Ce programme prétend seulement démontrer les bases de la construction d'un programme Java 3D. Il est aussi utilisé comme une base pour les programmes ultérieurs.

examples/HelloJava3D/HelloJava3Dc

Ce programme affiche un seul cube coloré en mouvement. Le cube tourne autour de son origine. Par conséquent, plusieurs faces du cube sont visibles dès que l'animation a lieu. Ce programme prétend démontrer les bases de la construction d'un programme animé Java 3D.

examples/HelloJava3D/HelloJava3Dd

Ce programme affiche un seul cube coloré ayant subit une transformation et en mouvement. Le cube tourne autour de son origine. Par conséquent, plusieurs faces du cube sont visibles dès que l'animation a lieu. Ce programme prétend démontrer les bases de la construction d'un programme animé Java 3D.

0.2.2 Géométrie

Le sous-répertoire `examples/Geometry` contient les programmes d'exemples du second chapitre du tutorial. Chacun de ces programmes décrit une partie de la création de la géométrie des objets visuels.

examples/Geometry/Axis.java

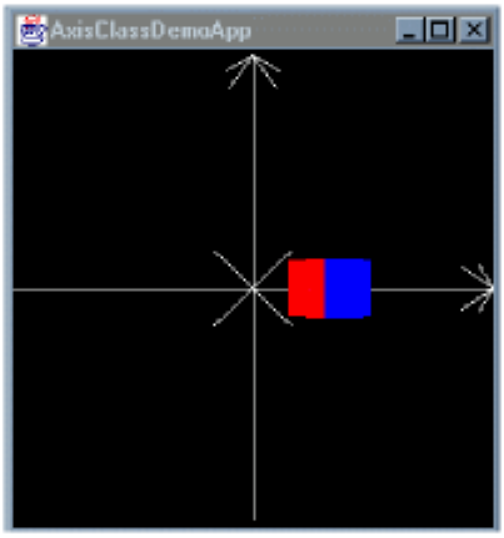
`Axis.java` définit une classe d'objet visuel utilisant un objet IndexedLineArray pour visualiser les axes. Le code de ce programme n'apparaît pas dans le texte du tutorial, étant une classe destinée à être disponible pour votre propre usage. Le programme `AxisClassDemoApp.java` utilise cette classe Axis (voir ci-dessous).

examples/Geometry/AxisApp.java

Ce programme affiche les axes pour présenter l'utilisation de l'objet LineArray.

examples/Geometry/AxisClassDemoApp.java

Dans ce programme un ColorCube est mis en orbite autour de l'origine. Le code ce programme n'apparaît pas dans le texte du tutorial. Il présente seulement l'usage de la classe `Axis.java` (voir page précédente).



examples/Geometry/ColorConstants.java

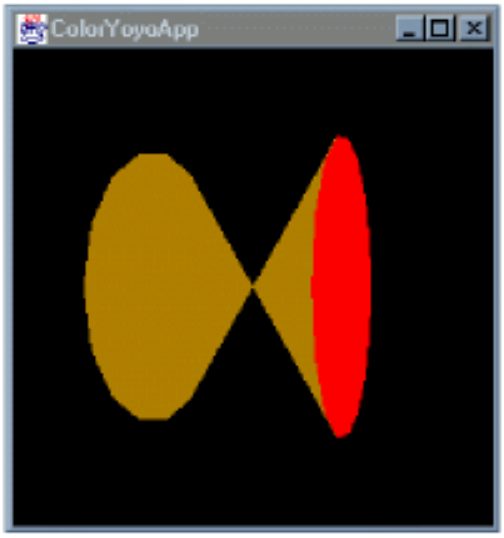
Ce code est un exemple de classe qui définit un nombre de couleurs constantes. L'application fait tourner le Yo-Yo autour de l'axe y pour présenter la géométrie.

examples/Geometry/ColorYoyoApp.java

Ce programme affiche un Yo-Yo utilisant quatre objets géométriques TriangleFanArray colorés. L'application fait tourner le Yo-Yo autour de l'axe y pour présenter la géométrie.

examples/Geometry/ConeYoyoApp.java

Ce programme affiche un Yo-Yo crée par deux objets Cone. L'apparence par défaut est utilisée. L'application fait tourner le Yo-Yo autour de l'axe y pour présenter la géométrie.



examples/Geometry/TwistStrip.java

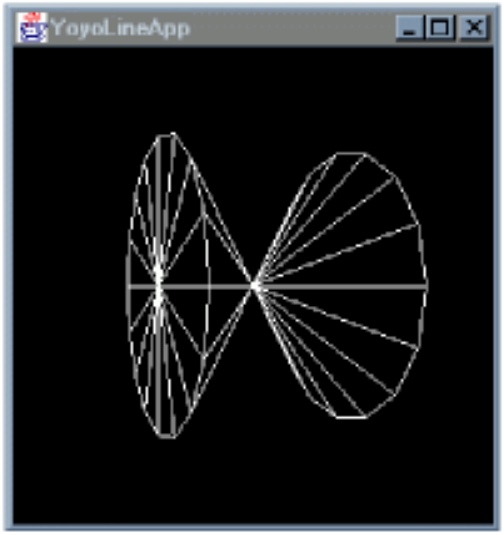
Ce programme affiche un ruban tordu comme exemple de l'usage du TriangleStripArray. Le programme du ruban tordu démontre aussi la sélection des faces à visualiser (cueillage des faces [culling]).

examples/Geometry/YoyoApp.java

Ce programme affiche un objet visuel Yo-Yo crée par quatre objets géométriques TriangleFanArray. Un objet Appearance par défaut est utilisé. L'application fait tourner le Yo-Yo autour de l'axe y pour présenter la géométrie.

examples/Geometry/YoyoLineApp.java

Ce programme affiche un objet TriangleFanArray avec un Appearance défini pour afficher seulement les lignes. L'application fait tourner le Yo-Yo autour de l'axe y pour présenter la géométrie.



examples/Geometry/YoyoPointApp.java

Ce programme affiche un objet TriangleFanArray avec une Appearance définie pour afficher seulement les points. L'application fait tourner le Yo-Yo autour de l'axe y pour présenter la géométrie.

0.2.3 Création de la géométrie

examples/easyContent/BackgroundApp.java

Cette application montre la définition d'une géométrie pour l'arrière plan d'un monde virtuel. La scène se compose d'une grille de lignes pour le sol, de PointArray pour des étoiles, et une LineArray pour une constellation. Les étoiles et la constellation sont en arrière plan. Le spectateur peut se déplacer dans la scène et percevoir le mouvement relatif entre le sol et les étoiles de l'arrière plan.

L'action (mouvement) est produite par la classe KeyNavigator (documentée au Chapitre 4) et par une limite d'application BoundingLeaf, ce qui produit une action sans limite dans le monde virtuel. Le BoundingLeaf est ajouté à la branche visualisation dans cette application.

examples/easyContent/GeomInfoApp.java

Cette application présente l'usage de la classe GeometryInfo pour créer une géométrie Java 3D construite par des polygones de façon arbitraire. Les classes Triangulator, Stripifier, et NormalGenerator sont utilisées pour convertir les polygones en une suite de triangles successifs pour spécifier la géométrie devant être traitée. Une vue en fil de fer de la géométrie peut être visualisée en rentrant la ligne de commande. Par exemple : `java GeomInfoApp -lines` au lancement du programme, montrant ainsi l'armature fil de fer sans les surfaces illuminées.

examples/easyContent/Text2Dapp.java

Un exemple simple de l'usage de l'objet Text2D pour introduire du texte dans un monde virtuel Java 3D. L'objet Text2D effectue une révolution dans le monde virtuel.

examples/easyContent/Text3Dapp.java

Un exemple simple de l'usage de l'objet Text3D pour introduire du texte dans un monde virtuel Java 3D. L'objet Text3D effectue une révolution dans le monde virtuel.

0.2.4 Interaction

Les programmes contenus dans le sous répertoire `examples/Interaction` correspondent aux thèmes abordés au Chapitre 4. Le sujet du chapitre étant la création et l'utilisation des comportements

examples/Interaction/DoorApp.java

Ce programme démontre l'usage de la méthode postId() et des objets WakeupOnBehaviorPost, WakeupCriterion pour coordonner les objets de comportement. Dans ce programme deux classes de comportement sont définies : OpenBehavior et CloseBehavior. Puis une instance de chaque classe de comportement est utilisée pour ouvrir et fermer une porte. En fait un ColorCube est utilisé comme une porte.

examples/Interaction /KeyNavigatorApp.java

Ce programme démontre l'usage de l'objet KeyNavigatorBehavior pour permettre une navigation dans le monde virtuel basé sur la frappe des touches du clavier. L'utilisateur peut se déplacer, par l'enfoncement des touches, en avant, en arrière, à droite, à gauche, vers le haut, et le bas et aussi effectuer une rotation vers la droite, la gauche, le haut et la bas.

examples/Interaction/MouseBehaviorApp.java

Ce programme montre comment les trois classes du MouseBehavior (MouseRotate, MouseTranslate, et MouseZoom) peuvent être combinées pour produire une variété d'interactions utilisant la souris. MouseRotateApp est une version simple qui n'emploie que la classe MouseRotate.

examples/Interaction/MouseNavigatorApp.java

Ce programme montre comment les classes du MouseBehavior (MouseRotate, MouseTranslate, et MouseZoom) peuvent être utilisées pour produire une navigation dans le monde virtuel basée sur les mouvements de la souris. L'utilisateur est capable d'effectuer des déplacements et des rotations en réponse aux pressions sur les boutons et les mouvements de la souris.

examples/Interaction/MousePickApp.java

Ce programme démontre les possibilités de l'interaction de pointage par l'usage de la classe PickRotateBehavior. L'utilisateur à la possibilité de pointer sur un objet visuel pour ainsi lui faire effectuer une rotation par des mouvements de la souris. C'est le contraire dans le programme MouseRotate2App qui présente comment l'utilisateur peut, sans pointage, être contraint à interagir avec les objets disponibles.

examples/Interaction/MouseRotateApp.java

L'utilisateur est capable d'effectuer une rotation pré-programmée de l'objet visuel grâce à la souris. MouseBehaviorApp est une version plus complexe de ce programme qui fournit la possibilité d'une translation et d'un zoom interactif en plus de la rotation. MouseRotate2App démontre la limite de cette classe.

examples/Interaction/MouseRotate2App.java

Une démonstration de l'usage de la classe MouseRotate pour produire une interaction avec des objets visuels spécifiques. Deux cubes répondent aux actions de l'utilisateur. Il n'est pas possible d'interagir sur un seul cube du programme. Ce programme démontre intentionnellement les limites de cette classe. MouseRotateApp est une version à un cube de ce programme.

examples/Interaction/PickCallbackApp.java

Il s'agit d'une modification de MousePickApp en y introduisant un effet-retour pour le comportement de pointage. L'utilisateur à la possibilité de pointer un objet visuel et le faire tourner avec des mouvements de la souris. Ce simple effet retour affiche un message dans la console. C'est une réponse à la question 6 dans la Partie des Tests Personnels.

examples/Interaction/SimpleBehaviorApp.java

Une classe de comportement simple est crée puis utilisée dans ce programme. Le comportement simple modifie un objet TransformGroup en réponse à la pression sur une touche. L'effet permet de tourner l'objet visuel par la frappe de touches du clavier. Le programme présente les bases de l'usage des comportements et comment écrire des classes de comportement personnalisé.

0.2.5 Animation

examples/Animation/AlphaApp.java

Ce programme illustre le lissage possible de l'onde produite par un objet Alpha. Trois objets visuels sont déplacés en utilisant trois objets PositionInterpolators et Alpha. Seul le paramètre IncreasingAlphaRampDuration de l'objet Alpha diffère pour les trois voiture-déterminants-alpha. Se référer à la Partie 5.2 et à la Figure 5-7 pour plus d'informations.

examples/Animation/BillboardApp.java

Ce programme illustre le comportement du Billboard produit par les objets de la classe Billboard. Un objet Billboard oriente un objet visuel de manière a ce qu'il soit toujours face à l'observateur. L'utilisateur de ce programme est libre de se déplacer en utilisant les touches du clavier. Se référer à la Partie 5.3 pour plus d'informations sur les applications et les API pour la classe Billboard.

examples/Animation/ClockApp.java

Ce programme utilise un objet Alpha et un RotationInterpolator pour faire tourner la façade d'une horloge analogique une fois par minute. La façade de l'horloge, définie dans `Clock.java`, est construite par un objet Alpha et deux RotationInterplotors. Le programme principal, de `ClockApp.java`, est une utilisation simple de l'usage d'un RotationInterpolator. La construction de l'horloge étant un peu plus complexe.

examples/Animation/InterpolatorApp.java

Ce programme illustre six classes de différents Interpolator dans une seule scène pour montrer la variété disponible de classes de déterminants.

examples/Animation/LODApp.java

Ce programme utilise un objet DistanceLOD pour représenter un objet visuel comme un des différents niveaux de représentation géométrique des variations de niveaux de détails. L'objet DistanceLOD choisit une des représentations géométriques sur la base de la distance entre l'objet visuel et l'observateur.

examples/Animation/MorphApp.java

Dans ce programme, une classe de comportement sur mesure anime un personnage en fil de fer qui marche, basé sur quatre objets GeometryArray formant les images clés.

examples/Animation/Morph3App.java

Dans ce programme, trois autres classes de comportement créent une animation basée sur, ou presque tous, les objets GeometryArray de la MorphApp. Ils sont nommés (de gauche à droite sur la Figure) «En place », « Tango », et « Interrompu ». L'animation complète n'est pas trop bonne. Bien sûr pour vraiment apprécier l'animation, vous devrez lancer le programme.

0.2.6Lumières

examples/light/LightsNPlanes.java

Ce programme rend une scène où trois surfaces sont éclairées par trois lumières différentes. Une lumière est directionnelle, l'autre est une lumière omnidirectionnelle, et la dernière est une lumière de projection (spot ou projecteur). Voir Figure 6-16.

examples/light/LitPlane.java

Ce programme est un exemple de base de l'utilisation des lumières. Il produit une scène avec une surface plane et une sphère. Voir la Figure 6-2.

examples/light/LitSphere.java

Ce programme est un exemple de base de l'utilisation des lumières. Il produit une scène avec une seule sphère. Voir la Figure 6-15 entre autres.

examples/light/LitTwist.java

Ce programme décrit l'éclairage d'un objet à deux faces (setBackFaceNormalsFlip()). Voir la Figure 6-21.

examples/light/LightScope.java

Ce programme décrit l'usage de la portée pour limiter l'influences des sources de lumière. Voir la Figure 6-25.

examples/light/LocalEyeApp.java

Ce programme illustre la différence entre un éclairage local et un éclairage infini. Voir Figure 6-29.

examples/light/ShadowApp.java

Ce programme explique la classe SimpleShadow. SimpleShadow cree des ombres polygonales pour des objets visuels simples dans certaines scènes. Voir la Ffigure 6-28.

examples/light/ShininessApp.java

Ce programme produit une scène statique de neuf sphères avec des propriétés de matériaux différentes. La seule différence de matériau étant la valeur de brillance [shininess]. Voir la Figure 6-20.

examples/light/SpotLightApp.java

Ce programme illustre par le rendu les différentes valeurs des parametres d’un projecteur cible [spot-light]. Voir la Figure 6-18.

0.2.7 Texture

Ces programmes d’exemples se trouvent déjà dans le fichier des exemples fournis avec le tutorial mais cette partie n’a pas encore été traduite (cela va arriver !).

0.3 (Annexe B) Documents de référence

0.3.1 Livres

Henry Sowizral, Kevin Rushforth, et Michael Deering, The Java 3D API Specification**,** Addison-Wesley, Reading, Mass., Décembre 1997. ISBN 0-201-32576-4

Ce livre décrit la version 1.0 de l'API Java 3D. Il y a quelques differences entre cette spécification et la version recente du produit. Il en fait une couverture compréhensible, mais n'est pas prévu comme un guide du programmeur.

Il est par ailleurs disponible en ligne à : http://sun.com/desktop/java3d

Il est aussi disponible en japonais : traduit par Yukio Andoh, Rika Takeuchi ; ISBN 4-7561-3017-8

Ken Arnold and James Gosling, The Java Programming Language**,** Addison-Wesley, Reading, Mass.

La référence Java.

David M. Geary, graphic JAVA Mastering the AWT**,** Sunsoft Press, 1997

Couvre complètement de l'AWT [Abstract Windowing Toolkit].

Foley, vanDam Feiner, and Hughes, Computer Graphics, **Addison-Wesley**

Ce livre est largement considéré comme « la bible du graphisme informatique ». Couvre de façon compréhensible tous les termes généraux du graphisme informatique, incluant les représentations de points, lignes, surfaces, et transformations. D'autres thèmes comme la projection, le texturage, le z-buffer et bien, bien d'autres.

OpenGL ARB, OpenGL Programming Guide**,** Addison-Wesley

Même s'il n'est pas en relation directe, ce livre procure de bonnes bases dans la programmation graphique par l'API OpenGL. Java 3D ressemble à OpenGL par de nombreuses de méthodes, et certaines des applications de Java 3D sont construites sur une implémentation OpenGL

0.3.2 Java 3D 1.1 peut être téléchargé depuis la page d'accueil de Java 3D

`http://java.sun.com/products/java-media/3D/`

Suivez le lien « java 3D 1.1.1 Implementation » vers la page `download.html`. Depuis cette page, vous pourrez télécharger la documentation pour les classes de l'API Java 3D.

0.3.3 Les pages web de Sun sur Java

Pour des informations supplémentaires, se référer aux pages de Sun Microsystems sur le World Wide Web.

`http://sun.com/desktop/java3d`

La page d'accueil du marketing de Java 3D, à cette page se trouvent de nombreux liens.

`http://java.sun.com/`

Le site web des Logiciels Java, avec les dernières informations sur la technologie Java, informations sur les produits, nouveautés, et caractéristiques.

`http://java.sun.com/products/jdk/1.2/`

Page du Produit JDK 1.2 et de téléchargement

`http://java.sun.com/docs`

La Plate-forme de Documentation sur Java fournit un accès au « livre blanc », au Tutorial de Java et à d'autres documents.

`http://developer.java.sun.com/`

Le site web de Connexion du Développeur Java (une inscription gratuite est requise) informations techniques additionnelles, nouveautés, et caractéristiques ; forums d'utilisateurs ; information sur le support, et bien plus.

`http://java.sun.com/products/`

Produits & APIs de Technologie Java

`http://www.sun.com/solaris/java/`

Kit de Développement Java pour Solaris - Version de Production

0.3.4 Autres Pages Web

Pour des informations supplémentaires, se référer à la page web de Java 3D pour des liens vers des ressources apparentées.

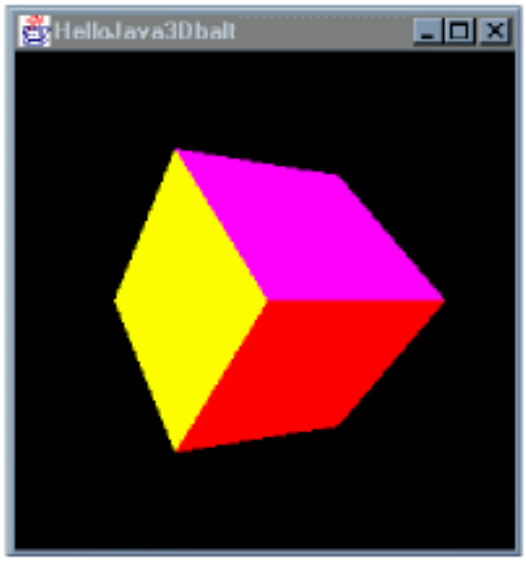
0.4 (Annexe C) Solutions choisies aux questions des Tests Personnel.

Chaque chapitre conclut sur une partie de Tests Personnels qui contient des questions destinées à tester et à augmenter la compréhension du lecteur sur le contenu du chapitre. Cette partie présente donc les réponses à quelques-unes de ces questions.

Note pour les questions relatives à la programmation. Comme n'importe quelle tâche de programmation, il y a de nombreuses réponses possibles aux questions de programmation. Cette partie fournit seulement une solution de programmation possible.

0.4.1 Réponses aux questions du Chapitre 1

1. Dans le programme HelloJava3Db, qui combine deux rotations dans un seul TransformGroup, quelle serait la différence si vous inversiez l'ordre de multiplication dans la définition de la rotation ? Modifiez le programme pour voir si votre réponse est correcte. Il y a deux lignes de code à modifier pour cela.



Réponse :

En général, l'orientation finale d'un objet dépend de l'ordre d'application des rotations. Il y a des cas où cet ordre des rotations ne modifie en rien l'orientation finale. Pour effectuer ce changement de l'ordre d'application des rotations, effectuez les deux modifications suivantes à HelloJava3Db.java. Puis compilez et lancez le programme modifié. Attention, si vous changez le nom du fichier, vous devrez également changer le nom de la classe principale du fichier. Par exemple, si vous changez le nom du fichier en HelloJava3Dbalt.java, la classe HelloJava3Db doit être modifiée en HelloJava3Dbalt, pour le nom du constructeur et l'appel à ce constructeur. Bien sûr, Les commentaires doivent également être modifiés pour refléter ces changements.

Modifier:

```
rotate.mul(tempRotate) ;
```

en:

```
tempRotate.mul(rotate) ;
```

Modifier:

```
TransformGroup objRotate = new TransformGroup(rotate) ;
```

en:

```
TransformGroup objRotate = new TransformGroup(tempRotate) ;
```

Après avoir effectué ces modifications, compilez, et exécutez le programme, l'image si dessus sera produite. Si vous comparez cette image à la Figure 1-12 du Chapitre 1, vous pouvez voir le résultat des modifications effectuées au programme.

2. Dans le programme HelloJava3Dd, quelle serait la différence si vous inversiez l'ordre des nœuds de transformation sous le ColorCube dans la branche volume ? Modifiez le programme pour voir si votre réponse est correcte.

Réponse :

Comme dans le programme précédent, la modification de cet ordre fait une différence. Comme dans la première question, seulement deux lignes de code doivent être rédiges pour tester le résultat des changements poses par la question.

3. Dans une recherche d'optimisation des performances, le programmeur veut rendre le graphe scénique moins lourd. Peut-on combiner les cibles des transformations de rotation et de révolution d'HelloJava3Dd dans un seul objet TransformGroup ?

Réponse :

C'est possible ; et ne demande pas beaucoup d'efforts. Un seul changement dans le graphe scénique augmente le nombre d'objets Shape3D, sans pour autant rendre le programme plus dur à écrire, lire, et à entretenir.

4. Déplacez le ColorCube d'une unité dans l'axe Y et tournez le cube. Vous pouvez utiliser HelloJava3Db comme point de départ. Le code qui suit la question montre l'instruction d'une transformation de translation. Essayez la transformation dans le sens inverse. Voyez-vous une différence au résultat ? Si oui, pourquoi ? Si non, pourquoi ? Essayez et comparez vos espérances au résultat actuel.

```
Transform3D translate = new Transform3D() ;
Vector3f vector = new Vector3f(0.0f, 1.0f, 0.0f) ;
translate.setTranslatiov(vector) ;
```

Réponse :

L'ordre des transformations fait une différence.

5. Dans HelloJava3Dc, la sphère de limitation d'effet a un rayon de 1 mètre. Cette valeur est-elle trop grande ou trop petite que ce qu'elle doit être ? Quelle est la plus petite valeur qui permettra une rotation du cube dans la visualisation ? Expérimentez sur le programme pour vérifier vos réponses. Les lignes de code qui suivent peuvent être utilisées pour définir une sphère de limitation d'effet. Dans ces lignes, l'objet Point3D spécifiant son centre, suivit du rayon.

```
BoundingSphere bounds =
     new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0);
```

Réponse :

L'objet BoundingSphere a seulement besoin d'un rayon de 0.8 (0.4 * 2.0). Son centre doit être à (0, 0, 0). La position de la BoundingSphere devra être transformée par n'importe quels objets TransformGroup situé

au-dessus d'elle sur le chemin du graphe.

6. Les programmes d'exemples donnent suffisamment d'informations pour assembler un univers virtuel avec plusieurs cubes colorés. Comment construiriez vous un tel graphe scnique ? Dans quelle partie du code cela doit-il être réalisé ?

Réponse :

Il y a de nombreuses manières d'ajouter un nouvel objet visuel à un univers virtuel. En voici quelques-unes unes :

- Ajouter un objet ColorCube comme un enfant au BranchGroup existant.
- Ajouter un objet ColorCube comme un enfant à un nouveau BranchGroup et ajouter ce BranchGroup à la Locale du SimpleUniverse.
- Ajouter un objet ColorCube comme un enfant au TransformGroup existant. Noter qu'ainsi deux ColorCube vont se trouver au même emplacement, un seul sera par conséquent visible.
- Ajouter un objet ColorCube comme un enfant à un nouvel objet TransformGroup et ajouter ce TransformGroup comme un enfant à la Locale du SimpleUniverse.
- Combiner les deux possibilités précédentes.

Un objet ColorCube ne peut pas être ajouté comme enfant de l'objet Locale.

0.4.2 Réponses aux questions du Chapitre 2

1. Essayez de créer par vous-même un nouveau Yo-Yo en utilisant deux cylindre à la place de deux cônes. Prenez comme base ConeYoyoApp.java, quels sont les modifications nécessaires ?

Réponse :

Tout ce que vous devez faire, c'est de remplacer les deux objets Cone par deux objets Cylinder

2. Un Yo-Yo de deux cylindres peut être crée avec deux objets ruban-carré et quatre objets éventail-triangulaire. Une autre solution est de réutiliser un seul ruban-carré et un seul éventail-triangulaire. La même approche peut être utilisée pour créer le Yo-Yo cône. Quels objets devront former cet objet visuel Yo-Yo ?

Réponse :

Dans chaque solution, l'objet visuel peut être défini par un Group avec deux objets TransformGroup, chacun ayant pour enfant un Shape3D. Chaque objet Shape3D se rapportant à la même référence géométrique de type NodeComponent (composant de nœud).

3. Le mode d'affichage par défaut des arêtes (mode de cueillage [culling]) est utilisé dans poyoLineApp.java et YoyoPointApp.java. Changez chacun ou un seul, de ces programmes pour n'afficher aucune arête, puis compilez et lancez le programme modifié. Quelle différence voyez-vous ?

Réponse :

Par défaut, les lignes (ou les points) sont affichées par leur cotés arrière (invisible). La mise hors fonction de cet affichage permet à toutes les lignes (points) d'être rendues dans toutes les orientations. Si vous n'avez pas déjà essayé, tournez les faces avant.

0.4.3 Réponses aux questions du Chapitre 3

1. En utilisant la vue fil de fer dans le programme, vous pouvez voir l'effet de la triangulation. Allez plus loin, changez l'ordre des polygones pour utiliser trois polygones un pour chaque côté, et un pour le toit, un pour le capot et les autres surfaces. Comment le Triangulator se comporte t-il avec cette surface ?

Réponse :

Le code pour la description des seuls polygones du capot, du toit, et la vitre avant et arrière est inclus dans le code de l'application d'exemple. Voyez le code source pour plus de détails. Cependant, il y a trois autres lignes de code qui doivent être changées pour utiliser cette forme alternative de polygone. Malheureusement, le Triangulator ne triangule pas correctement le polygone du coté. Vous devez donc essayer de rediviser le grand polygone pour que cela fonctionne. Cette leçon doit vous donner une vision du mode de fonctionnement des surfaces, en utilisant GeometryInfo.

2. Le code pour rendre l'objet Text2D visible des deux cotés est inclus dans `Text2DApp.java`. Vous pouvez enlever les caractères // de commentaire de ce code, puis le compiler et l'exécuter. D'autres expérimentations se trouvent dans ce programme, comme l'application de la texture de l'objet Text2D sur d'autres objets visuels. Par exemple, essayez d'ajouter une géométrie primitive et appliquez la texture sur celui-ci. Bien sur, vous devrez attendre la lecture du chapitre 7 pour cet exercice.

Réponse :

Le code pour rendre pour rendre visible l'objet Text2D des deux cotés est encastré dans les commentaires du programme d'exemple Text2DApp.java. Voyez le code source pour plus de détails.

3. En prenant `Text3DApp.java` comme point de départ, expérimentez les différents alignements et placements du point de référence. D'autres expérimentations peuvent être le changement d'apparence de l'objet Text3D.

Réponse :

Essayez seulement les différentes positions et observez les differences.

4. Lancez le programme d'exemple `BackgroundApp.java`, si vous vous déplacez trop loin de l'origine du monde virtuel, l'arrière plan disparaît. Pourquoi cela se produit-il ? Si vous ajoutez un autre Background à la BackgroundApp, quelle conséquence cela aurait-il ?

Réponse :

La mise en place d'un arrière plan est contrôlée par un ApplicationBounds (ou applicationBoundingLeaf) pour l'arrière-plan. Dans cet exemple, l'arrière-plan possède une sphère ApplicationBounds centrée à l'origine avec un rayon de 10,000.

Si un autre arrière-plan était ajouté, le résultat dépendrait de l'ApplicationBounds spécifié pour ce nouvel arrière plan. Dans tous les cas, un seul arrière- plan sera chaque fois rendu pour n'importe quelle scène. La sélection de l'arrière plan dépend de l'ApplicationBounds et de la position du spectateur dans le monde virtuel.

0.4.4 Réponses aux questions du Chapitre 4

1. Écrivez un comportement réalisant une commande sur mesure, le déplacement des objets visuels vers la gauche (et la droite) quand la touche fléchée gauche (et droite) est enfoncée. Puis utilisez cette classe dans une application semblable à simpleBehaviorApp.java. Bien sûr, vous pouvez utiliser SimpleBehaviorApp.java comme base pour les deux comportements sur mesures et son application. Que se passera t-il si l'objet ColorCube sort de la vue ? Comment corrigerez-vous le problème ?

Réponse:

Quand le cube se déplace à une distance suffisante, la surface planifiée du comportement ne coïncide plus avec l'objet visuel. Comme l'objet de limitation est associé au comportement, et que le comportement ne se déplace pas avec l'objet, ils seront en fin de compte séparés à l'endroit où le comportement ne sera plus actif quand le cube deviendra invisible. Inversement, le comportement sera inactif quand le cube sera visible. Donc, si vous ajoutez une aptitude de navigation au programme, voir le cube ne signifiera pas forcement que vous pourrez interagir avec lui.

Il y a de nombreuses manières de modifier le programme pour que la position de cube et de la surface planifiée du comportement qui coïncide. L'une d'entre elles est de rendre le comportement enfant de l'objet groupe de transformation qui permet de déplacer cet objet visuel. L'autre demande l'usage d'un objet BoundingLeaf. Voir le Chapitre 3 pour des informations supplémentaires sur la classe BoundingLeaf.

2. Dans SimpleBehaviorApp, la rotation est calculée par une variable d'angle de type double. Cette variable d'angle est utilisée pour établir la rotation d'un objet Transform3D lequel determine la transformation d'un TransformGroup. Une alternative voudrait qu'on élimine cette variable d'angle en utilisant un seul objet Transform3D pour contrôler l'accroissement de l'angle. Il y a deux variations à cette approche, une lirait la transformation actuelle du TransformGroup et la multiplierait ensuite, une autre stockerait la transformation dans un objet partiel Transform3D. Dans tous les cas, la nouvelle rotation est trouvée par la multiplication du précédent par le Transform3D qui contient l'addition de la rotation. Quel problème peut-il se produire avec cette alternative ? Quelle amélioration peut-on apporter à cette approche ?

Réponse:

Des rotations successives (ou des transformations d'un autre type) peuvent être réalisées par l'usage de multiplications successives des transformations. Le problème se situe dans la perte de précision par les multiplications répétées. Il prend beaucoup d'itérations, mais par la suite l'erreur s'accumulera et aura comme conséquence des effets étranges au rendu.

3. Changez la condition de déclenchement dans la classe de SimpleBehavior en un **newElapsedFrame(0)**. Compilez et exécutez le programme modifié. Notez le résultat. Changez le code pour retirer le problème de brûlure de mémoire de la classe. Puis recompilez et exécutez le programme ainsi fixé.

Réponse :

Le programme déclenchera sur chaque trame (et est donc maintenant une application d'animation). En conséquence, un nouvel objet est créé sur chaque trame. Puisque des objets sont créés à une vitesse assez rapide et pas réutilisés, c'est un cas de brûlure de mémoire. Le rendu fera une pause quand le collecteur d'ordures (garbage collector) se lancera pour nettoyer la mémoire.

4. Changez les limites de planification pour l'objet de KeyNavigatorBehavior en quelque chose de plus petit (par exemple, une limite sphérique avec un rayon de 10), puis re-exécutez l'application. Que se produit-il quand vous vous déplacez au-delà des nouvelles limites ? Convertissez les limites de planification de la KeyNavigatorApp en application universelle de sorte que vous ne puissiez plus vous coincer au bord du monde. Voir le Chapitre 3 pour plus d'information sur les nœuds BoundingLeaf.

Réponse :

Quand l'utilisateur se déplace au-delà des limites de planification, le comportement de navigation devient inactif et peut donc plus ne répondre aux frappes (ou à tout autre déclenchement). La visualisation se bloque et l'utilisateur doit quitter le programme.

5. Utilisez le KeyNavigatorBehavior avec un TransformGroup au-dessus d'un objet visuel dans la branche volume du graphe. Qu'est ce qui ce produit ?

Réponse:

L'effet produit est de déplacer l'objet visuel, pas le spectateur. L'action se situe ainsi en arrière de l'application normale de ce comportement.

6. Étendez le comportement de pointage dans le MousePickApp en fournissant un rappel de service. Vous pouvez commencer par produire simplement une chaîne de caractères des textes («picking») dans la console. Vous pouvez également devenir plus ambitieux et lire les données de l'utilisateur depuis le groupe de transformation cible ou enregistrer le déplacement et/ou les rotations du groupe de transformation cible. Avec les capacités appropriées, vous pouvez également accéder aux enfants de l'objet de TransformGroup.

Réponse:

Voir `examples/Interaction/PickingCallbackApp.java`.

0.4.5 Réponses aux questions du Chapitre 5

1. Le programme d'exemple InterpolatorApp utilise six objets déterminants différents. Chacun des objets déterminants se rapporte au même objet d'alpha. Le résultat doit coordonner tous les déterminants. Quel serait le résultat si chaque objet avait son propre objet d'alpha ? Comment pourriez-vous changer la synchronisation ?

Réponse :

Si chaque déterminant utilisait un objet alpha différent, les déterminants seraient de toute façon tous synchronisées. À chaque objet alpha est assigné une heure de départ identique quand il est créé. Si vous vouliez que les objets alpha aient lieu hors période, il faudrait changer l'heure de départ ou assigner une durée de période différente.

2. Si la lumière dans InterpolatorApp est changée en `Vector3f(-0.7f, -0.7f, 0.0f)` que se produit-il ? Pourquoi ?

Réponse :

Le corps de la voiture disparaît ; cependant, les roues sont encore visibles et changent de couleur comme avant. Pourquoi les différentes pièces de la voiture sont-elles rendues différemment ? Si vous changez l'arrière plan dans une couleur différente, vous verrez que le corps de la voiture est toujours là mais il est complètement blanc. C'est le résultat de la réflexion spéculaire complète du corps de la voiture. La combinaison de la direction de la lumière et des normales du corps de voiture reflètent la source lumineuse comme le fait un miroir. C'est un problème qui concerne le chapitre six, mais est une tête-de-ligne potentielle pour des lecteurs du Chapitre cinq.

3. Pourquoi y a-t-il une distance moins importante pour les objets visuels spécifiés par un objet DistanceLOD ?

Réponse:

Par sa conception le seuil commence à utiliser le premier enfant dont l'objet(s) cible est tourné à zéro. Ce seuil n'est pas indiqué par l'utilisateur. Les distances de seuil indiquées par l'utilisateur sont les distances auxquelles les enfants restants doivent être utilisés. Par conséquent, il y a autant de distances minimums spécifiées qu'il y a d'enfants.

4. Dans MorphApp, il y a quatre frames dont deux semblent être les copies des deux autres. Pourquoi quatre frames sont-elles nécessaires ? Posé autrement, quel aspect prendrait l'animation avec seulement deux frames ?

Réponse:

Puisque Morph procède à l'animation entre les frames sur la base de l'ordre des sommets, avec seulement deux frames le mouvement s'effectuera par un aller-retour entre les deux frames. Puisque le «pied » de l'animation n'apparaissent jamais au même endroit dans chacune des frames clé, quatre frames sont nécessaires. La différence entre les frames qui paraissent les mêmes est l'ordre des sommets. Il serait possible d'animer la marche avec deux trames si dans une des frames un pied est derrière l'autre (du point de vue du spectateur). Naturellement, ceci fonctionnerait seulement avec les modèles2d.

5. En utilisant un objet de morphing, le même nombre de sommets sont utilisés. Comment pouvez-vous adapter les modèles géométriques qui ont des nombres différents de sommets ?

Réponse:

Dans une géométrie, avec le plus petit nombre de sommets le compte de sommet doit être augmenté jusqu'à être égal à celui de la géométrie plus grande. Les nouveaux sommets peuvent être redondants ou internes sur une surface.

0.4.6 Réponses aux questions du chapitre 6

1. Ajoutez un DirectionalLight vert se dirigeant vers la `LitSphereApp` pour illustrer un mélange de couleur additive avec les trois couleurs primaires. N'oubliez pas d'ajouter la lumière au graphe et fixer la limite d'influence pour le nouvel objet source de lumière. Quelles sont les deux couleurs primaires qui formeront du jaune ?

Réponse:

Dans le système positif des couleurs, vert et rouge s'unissent (s'ajoutent) pour former le jaune. Ce n'est pas le résultat du mélange pictural du vert et du rouge, mais de la lumière verte et rouge. Le mélange de la peinture rouge et verte aura probablement comme résultat une certaine nuance de brun (selon les caractéristiques de la peinture).

2. Afin de mieux comprendre le sujet de l'interaction entre les couleurs des matériaux et les couleurs des lumières, créez une scène avec des objets visuels rouges, verts, et bleus. Allumez les objets avec une lumière simple de couleur simple. Que voyez-vous ? Vous pouvez utiliser `LitSpereApp.java` ou MaterialApp.java comme point de départ.

Réponse:

Dans la lumière monochromatique (rouge, vert pur, et bleu) seuls les objets visuels ayant une couleur matiére de la couleur de la lumière sont visibles. Contrairement, dans une lumière monochromatique, les

objets visuels ayant une couleur de matériau absente de la couleur de la lumière sont invisibles.

Les résultats que vous obtenez de votre programme dépendront de votre programmation. Si vous paramètrez seulement la couleur diffuse des objets visuels, alors le point culminant spéculaire apparaîtra dans la couleur de la lumière (le blanc par défaut).

3. Utilisez LightScopeApp.java comme point de départ (voir Partie 6.6.2), modifiez le programme pour créer les ombres de la boite éclairée par l'unique usage de la portée

Réponse:

Un groupe supplémentaire de type nœud est nécessaire. Il est placé entre le litBoxTG et l'objet litBox. Il référence la portée de ce nouveau groupe de nœuds. Vous devez également changer la couleur diffuse du matériau en blanc. L'image rendue de la scène est différente puisque l'ombre est définie par les autres lumières de la scène.

0.4.7 Réponses aux questions du chapitre 7

Cette section est intentionnellement laissée (presque) blanche. La suite doit arriver ...

0.5 Lexique

A

actif	Dit d'un objet de comportement qui est capable de recevoir un stimulus de réveil et donc de s'exécuter. Un objet de comportement est actif quand sa limite de planification intersecte avec le volume d'activation du ViewPlatform. Quand un comportement est inactif, Java 3D ignore les critères de réveil de ce comportement. Voir également *volume d'activation* et *comportement.*
agrafage [stitching]	Quand la même géométrie est rendue dans une image en fil de fer et en polygones pleins (de différentes couleurs), par défaut la profondeur des pixels rendus pour chacun ne correspondra pas et ainsi l'image en fil de fer semblera rentrer et sortir de la surface comme des fils de la surface d'un vêtement. Voir la Partie 2.6.3. PolygonAttributes dans le Module 1 pour des informations supplémentaires.
alpha	Dans le graphisme informatique, le terme «alpha» se réfère normalement à la transparence. Dans Java 3D alpha se réfère aussi a la transparence, mais aussi à la classe Alpha, ou a une valeur alpha. Ce qui peut être à l'origine de confusions pour les programmeurs graphiques expérimentés. Voir aussi *Classe Alpha*, *valeur alpha*, *RGBA* et *transparence.*
ancêtres (d'un objet)	Tous les objets d'un graphe sont les enfants d'un objet précis et cet objet est l'ancêtre de tous les enfants. Voir aussi *objet*, *graphe* et *relation parent-enfant.*
angle de faux pli	Dans une surface, l'angle minimum entre les surfaces de triangles adjecents doit être un angle discontinu. Quand un pli est détecté par le NormalGenerator il produit d'autres normales pour les sommets.
animation	Le changement automatique du contenu visuel (e.g., changements basés seulement sur le temps). Dans ce tutorial, les animations sont conduites par le passage du temps, les changements sur la plate-forme de visualisation, et peut être les non-actions de l'utilisateur. L'animation est le sujet du Chapitre 5. Voir également *interaction.*
anti-crénelage [anti-aliasing]	Un procédé de lissage de l'affichage des points ou des lignes qui autrement apparaîtraient en dents de scie. Voir aussi *dents de scie*, *anti-crénelage de ligne*, *anti-crénelage de point.*
anti-crénelage de ligne	Un attribut d'apparence qui, quand il est permit, provoque au rendu l'application de l'anti-crénelage aux lignes qui devant être rendues. Voir aussi *anti-crénelage* et *rendu.*
anti-crénelage du point	Un attribut d'apparence qui, lorsqu'il est attribué, provoque au rendu l'application de l'anti-crénelage des points devant être rendus. Voir aussi *anti-crénelage*, *rendu* et *dents de scie*.

API	Application Programming Interface. Terme général se référant à une collection de classes ou de méthodes (ou procédures) qui forment une interface de programmation pour un systeme informatique. Les classes de l'API Java 3D offrent une interface au programmeur pour le rendu en Java 3D.
aptitudes [capacities]	Dans le sens de Java 3D, les accès aux parametres d'un objet vivant sont contrôlés par les aptitudes. Par exemple, la transformation d'un TransformGroup vivant ne peut pas être changé sans une aptitude ayant été déterminée pour cette instance de TransformGroup avant qu'il soit né. Voir aussi *vivant* et *TransformGroup.*

B

bandification [stripification]	Organisation des triangles sur une bande pour la rapidité du rendu.
Billboard	L'orientation automatique d'un polygone dans une direction orthogonale au spectateur, pour que la face avant, seulement, du polygone soit visible. Typiquement le polygone est texturé avec une image.
Behavior	Voir *Comportement.*
branche du graphe [(branch graph]	La partie du graphe scènique qui prend racine dans un objet BranchGroup. Voir aussi *BranchGroup, graphe scénique, branche volume du graphe et branche visualisation du graphe.*
branche visuelle du graphe	La partie du graphe scénique contenant un objet View. Les parametres de l'environement visuel (e.g., stéréo) et les parametres physiques (e.g., position physique du spectateur) sont définis par la branche visuelle du graphe.
branche volume du graphe [content branch graph]	La partie du graphe scénique qui contient les objets visuels. Voir aussi *volume* et *objets visuels.*
brillance [shininess]	La spécification de la brillance de la surface d'un matériau. La valeur (dans la gamme de 1.0 à 128.0) est utilisée pour calculer la réflexion spéculaire d'une lumière sur un objet visuel. La valeur la plus haute étant la réflexion spéculaire la plus concentrée. Voir aussi *réflexion spéculaire* et *propriétés des matériaux*.

C

capteur	Le concept abstrait d'un périphérique d'entrée comme un joystick, ... [tracker, or data glove].
carré	Un raccourci pour quadrilatère. Un polygone à quatre cotés.
champ [field]	Endroit où l'on déclare les variables, donnée non-statique par défaut.
chargeur [loader]	Une classe utilitaire Java 3D qui crée les éléments du graphe scénique depuis un fichier. Les Chargeurs existent pour les formats de fichiers 3D les plus communs.

chemin de la branche du graphe scénique
: Le chemin depuis un objet Locale ou un nœud intérieur, vers une terminaison du graphe de la scène. SceneGraphPath est une classe utilisée pour le pointage. Voir Chapitre 4 pour plus d'information sur cette classe.

claquage de mémoire
: L'estimation entre la mémoire allouée et le ramassage des déchets pendant qu'une application tourne. Le claquage de mémoire est souvent causé par la création et la destruction d'objets inutiles. Le claquage de mémoire peut affecter défavorablement la performance du rendu dans certains environnements d'utilisations. La mémoire se claquant avec les comportements.

classe
: L'énumération d'un jeu de valeurs et d'un jeu d'opérations. Une classe est analogue à un type dans un langage de procédures comme C. Voir aussi *objet.*

classe Alpha
: Un objet d'une classe Alpha crée une fonction de variation temporelle de valeurs alpha. Normalement un objet Alpha est utilisé avec des objets pour créer une animation basée sur le temps. Voir aussi *valeur alpha, animation* et *interaction.*

classe Appearance
: Les instances d'Appearance définissent l'apparence d'un objet Shape3D. Les objets Appearance contiennent les objets AppearanceAttribute. Voir aussi *AppearanceAttributes* et *Shape3D.*

classe Behavior (comportement)
: Une classe de javax.media.j3d utilisée pour déterminer quelques caractéristiques, comme la position, l'orientation, ou la taille, qui changent à l'exécution. Les classes Behavior sont utilisées dans la création d'animations mais aussi bien dans des programmes interactifs. Voir aussi *comportement, Interpolator* et *classe Alpha.*

classe BoundingBox
: Les instances de BoundingBox définissent des limites par une boite alignée sur les axes. Voir aussi *volume de délimitation.*

classe BoundingSphere
: Les instances de BoundingSphere définissent des limites de forme sphérique. Voir aussi *volume de délimitation.*

classe Bounds (limite)
: Une classe abstraite pour spécifier un volume de délimitation pour lequel quelque chose est déterminé. Les Bounds sont utilisés avec des comportements, arrière-plans, sons, et d'autres dispositifs de Java 3D. Les Bounds peuvent être définis par un BoundingBox, BoundingSphere ou un BoundingPolytope. Voir aussi *volume de délimitation, polytope,* et *région de planification.*

classe BranchGroup
: Les instances de BranchGroup sont les racines de branches individuelles du graphe scénique, appelés branches du graphe. Un BranchGroup peut avoir de nombreux enfants que sont les objets Group et/ou Leaf. Un BranchGroup est le seul objet qui puisse être inséré à un objet Locale. Voir aussi *graphe scénique, branche du graphe,* et *Locale.*

classe de complaisance [convenience class]	Une classe qui étend une classe racine dans l'intention de rendre une classe (ou des classes) racine plus simple d'utilisation. Les classes de complaisance se trouvent dans le package com.sun.j3d.*.
classe Group	Group est une classe abstraite. Des instances de sous-classes Group sont utilisées pour créer le graphe scénique. Un Group est un objet du graphe scénique dont la fonction primaire est d'être le parent des autres objets du graphe scénique (objets Leaf ou autres objets Group).Voir aussi *BranchGroup* et *TransformGroup.*
classe Locale	Les instances de Locale produisent un point de repère dans l'univers virtuel. La position de tous les objets visuels est relative à un objet Locale. Les objets Locale sont les seuls objets se référençant aux objets VirtualUniverse. Les objets BranchGroup sont les seuls enfants d'un objet Locale. Voir aussi *BranchGroup, classe VirtualUniverse, objet visuel,* et *branche du graphe.*
classe SpotLight	Une classe de source de lumière qui contient des valeurs de position, direction, d'angle d'ouverture et de concentration. Voir le Chapitre 6 pour plus d'information. Voir aussi *concentration.*
classe TransformGroup	Une sous-classe de Group qui applique aussi à ses nœuds enfants une transformation. Voir aussi *transformation.*
classe utilitaire [utility class]	Une classe du package com.sun.j3d.utils, qui enrichit les classes racines par d'autres possibilités de programmation.
classe View	L'objet View est l'objet central pour coordonner tous les aspects d'une vue incluant les Canvas(es). Java 3D supporte des vues multiples simultanées.
classe VirtualUniverse	La classe racine de Java 3D. Une instance de VirtualUniverse doit être à la racine du graphe scénique.
classes Attributes	Les instances d'Attributes définissent les attributs d'apparence spécifiques d'un objet Shape3D. Il y a de nombreuses classes Attributes. Les objets Appearance incluent des objets Attribute. Voir également *classe Appearance* et *Shape3D.*
classes Color*	Un jeu de classes utilisé pour représenter une simple valeur de couleur. Les classes Color3b, Color3f, Color4b, et Color4f.
classes Color3*	Un jeu de classes utilisé pour représenter une simple valeur de couleur RGB. Les classes Color3b, Color3f. Voir aussi *Color** et *RGB.*
classes Color4*	Un jeu de classes utilisé pour représenter une simple valeur de couleur RGBA. Les classes Color4b, Color4f. Voir aussi *Color** et *RGBA.*
classes Geometry	Le lieu de location des classes de ce type formant ainsi un super-classe de géométries comme GeometryArray, et Text3D.
classes Point*	Point* se réfère à un, ou à tous les nombres de classes utilisées pour représenter des points en Java 3D. Consulter une référence pour les classes Point2f, Point3f, Point3d, ...

classes Tuple*	Un jeu de classes défini dans le package javax.vecmath utilisé pour représenter des tuples (tuple = triple; contient trois chiffres pouvant etre les coordonnées d'un point dans un espace n). Les sept classes individuelles sont Tuple2f, Tuple3b, Tuple3d, Tuple3f, Tuple4b, Tuple4f, et Tuple4d. Ces classes sont les super-classes des classes Color*, Point* et Vector* (parmi d'autres).
compile	Dans le sens Java 3D, compile() est une méthode d'un BranchGroup pour fournir une amélioration de l'exécution dans la branche du graphe fixée sur le BranchGroup. Voir aussi *BranchGroup* et *branche du graphe.*
comportement [Behavior]	La modification de quelques caractéristiques, comme la position, l'orientation, la taille, la couleur ou une combinaison de ceux-ci, d'objets visuels à l'exécution. Voir aussi *classe Behavior.*
concentration	Un paramètre d'un objet projecteur de lumière qui détermine l'étendue de la lumière depuis la source d'éclairage. Voir aussi *projecteur de lumière.*
condition de réveil	La combinaison des critères de réveil qui définissent la condition de déclenchement d'un objet de comportement. Java 3D appelle la méthode processStimulus de l'objet de comportement en réponse à l'événement de condition de réveil pour un objet de comportement actif. WakeupCondition est une classe présentée dans le Chapitre 4. Voir aussi *comportement, processStimulus* et *critère de réveil.*
constructeur	Pseudo-méthode créant un nouvel objet. Les constructeurs sont des instances qui portent le même nom que leur classe. Les constructeurs sont appelés à l'aide du mot clé `new`.
couleur ambiante (d'un matériau)	Partie des propriétés matérielles d'un objet visuel. La couleur ambiante d'un matériau et une lumière ambiante dans la scène produisent une réflexion ambiante.
couleur d'émission (d'un matériau)	Une couleur spécifiée pour amener un objet à une auto-illumination. Cet objet n'est pas une source de lumière et n'éclaire pas les autres objets, mais il est visible en l'absence de sources de lumières. Voir aussi *propriétés des matériaux.*
crénelage [aliasing]	L'aspect dents de scie [jaggies] des lignes et des courbes. Le résultat d'un échantillonnage dans une fonction continue. Voir *dents de scie* et *anti-crénelage* [antialiasing].
critère de réveil	Une combinaison d'objets de critères de réveil forme une condition de réveil pour un objet de comportement. Plusieurs des critères de réveil sont possibles, comprenant des AWTEvents, des collisions, des activateurs de comportement, le passage du temps, et un nombre bien spécifique de frames devant être dessinées. WakeupCriterion est une classe présentée dans le Chapitre 4. Voir aussi *condition de réveil.*
cueillir ou sélectionner [cul]	Pour retirer quelque chose non valable ou non nécessaire. Voir aussi *cueillir une face.*

cueillir une face [cull face]	Une face qui ne doit pas être rendue. Une face cueillie est spécifiée pour enlever du rendu une face non nécessaire, par exemple, une face intérieure ou une surface fermée. Cueillir des faces est indiqué pour les faces avant ou les faces arrières. Une face peut etre cueillie pour optimiser l'exécution. Voir également *face avant.*
D	
DAG.	Graphe Orienté Acyclique. Voir *Directed Acyclic Graph.*
demi-espace	L'espace sur un coté d'un plan. Un plan divise toute sa surface en deux parties, une moitié de chaque coté du plan.
dents de scies [jaggies]	Le terme technique pour le crénelage qui apparaît sur les points et les lignes et polygones au rendu. Ce crénelage apparaît quand les pixels utilisées sortent individuellement du tracé. Voir aussi *anti-crénelage.*
désactivation	Quand la région planifiée d'un comportement et le volume d'activation de la ViewPlatform ne se croisent plus, le comportement est désactivé. Comme un comportement ne peut pas etre mit en éveil lorsqu'il est inactif, un objet de comportement désactivé ne produira rien. Voir aussi *actif* et *comportement.*
déterminant [interpolator]	Se réfère à une des nombreuses classes sous-classées par la classe (Interpolator), ou à un objet d'une de ces classes. Les objets Interpolator produisent des animations dans un monde virtuel Java 3D par la variation de plusieurs paramètres d'un objet cible du graphe scénique. Les changements effectués par le déterminant sont définis par le ou les parametres du déterminant et de l'objet Alpha dirigeant ce déterminant. Voir aussi *classe Alpha, comportement,* et *objet cible.*
détermination [interpolation]	Le calcul d'une valeur, ou d'un jeu de valeurs, basé sur la valeur d'un nombre(integer) simple. Quelquefois la valeur dérivée est imbriquée entre deux valeurs, ou deux jeux de valeurs ; d'autres fois, la valeur dérivée est le résultat d'une formule. Spécifiquement, il y a un jeu de classes déterminantes utiles pour l'animation. Voir *Partie 4.1* et *Chapitre 5 pour plus de détails.*
Directed Acyclic Graph	Une structure composée d'éléments et d'arcs directeurs en laquelle aucun cycle n'est formé. Dans le monde de Java 3D, les éléments du DAG sont les objets Group et Leaf (lit. bourgeon ou feuille, ici les terminaisons), et les arcs sont les rapports de parent-enfant entre les objets Group et les Leaf. Former une structure sans cycles signifie qu'aucun élément ne peut être son propre parent.
E	
échelle	Le fait de changer la forme d'un objet visuel par la transformation de chaques sommets de l'objet. Voir aussi *transformation.*
exception	Un bug à l'exécution qui stoppe net l'exécution à sa détection. L'API Java 3D définis de nombreuses exceptions.

F

face avant — La face d'un polygone dont les sommets sont dans le sens horaire. Un procédé mnémotechnique pour s'en souvenir est d'appliquer la règle de la main droite. Voir aussi *règle de la main droite* et *ceuillir les faces.*

feuille de limite de planification [scheduling bounding leaf] — Une alternative aux limites de planifications. Voir aussi *limite de planification.*

flag (drapeau ou marqueur) — Valeur boléene dans un programme representant un état à l'intérieur d'un ensemble d'états.

frame clé [key frame] — Un terme utilisé dans l'animation traditionnelle et informatique pour une image de l'animation sur laquelle les autres sont basées. Le processus de la création des frames intermédiaires est appelé « in-betweening ». Les animations faites de frames clés et d'images intermédiaires sont appelées animations par frame clé. Un objet Morph peut être utilisé pour fabriquer des frames clés en Java 3D.

frustum — Fragment, morceau voir aussi *frustum de visualisation*

frustum de la visualisation — Le volume de visualisation de la forme d'une pyramide tronquée qui défini tout ce que le spectateur voit. Les objets qui ne se trouvent pas dans le frustum de la visualisation ne sont pas visibles. Les objets qui intersectent avec les limites du frustum de la visualisation sont coupés. Voir la Figure 1-9 pour une illustration. Voir aussi *spectateur* et *objet visuel.*

G

garbage collector — libérateur de mémoire. Détecteur et libérateur automatique de la mémoire qui n'est plus utilisée

get* ou get-method — *acquisition ou méthode d'acquisition*
Une méthode d'une classe qui reçoit un champ de valeur d'un objet. Voir aussi *set**.

glyphe — L'image d'un caractère dans une police de caractère. Voir aussi *police de caractère* et *style de police.*

graphe scénique — La structure Java 3D qui détermine les objets visuels et les paramètres de visualisation dans un univers virtuel.

H

I

illumination [shade] — **n.** La couleur d'un pixel ou d'un sommet qui est le résultat de l'application d'un mode d'éclairage (pour un sommet) ou du mode d'illumination (pour un pixel).

v. Calcul la couleur d'un sommet ou d'un pixel par l'application d'un mode d'éclairage (pour un sommet) ou du mode d'illumination (pour un pixel).

illumination Flat (modèle de Lambert)
: Illuminer ou donner du volume un objet avec la couleur de chacun de ces vecteurs sans interpolation. Voir aussi *modes d'illumination*.

illumination Gouraud
: Ombrage lissé d'un objet par une interpolation tri-lineaire des valeurs de couleur aux sommets des objets. Voir aussi *ombrage flat, interpolation tri-lineaire.*

image plate ou tableau
: Le rectangle imaginaire dans l'univers virtuel sur lequel est projeté. Voir la Figure 1-9 pour exemple. Voir aussi *frustum de visualisation*

influence (d'une lumière)
: La région (volume) avec laquelle le volume de limitation d'un objet visuel doit intercepter pour cet objet soit éclairé par la source de lumière. Voir aussi *limite.*

instance (d'une classe)
: Une instance d'une classe est un objet spécifique, individuel construit par la classe nommée.

interaction
: Changement d'état d'un monde virtuel en réponse aux actions de l'utilisateur. C'est en contraste avec l'animation, laquelle est définie comme un changement dans le monde virtuel non causé par une action de l'utilisateur. L'interaction est le sujet du Chapitre 4.
Voir aussi *animation.*

interface
: Semblable à une classe abstraite, une interface définit des méthodes qui seront exécutées par les autres classes quelque soient leurs emplacement dans la hiérarchie. Aucun constructeur n'est définit dans une interface et toutes les méthodes définies dans une interface sont abstraites.

interpolation tri-linéaire
: L'utilisation d'interpolations tri-linéaires pour arriver à une valeur. Cette technique est utilisée pour calculer la valeur d'illumination pour un pixel depuis la valeur des sommets en illumination Gouraud. Voir aussi *illumination Gouraud* et *illumination Flat.*

J

Java 2D
: Une API pour le graphisme en 2D.

K

K Computing
: La société de formation et de consulting qui a développé ce tutorial. Voir aussi à : http://www.kcomputing.com

L

limite d'application [application bounds]
: La région pour laquelle quelque chose est appliqué. Les arrière-plans [Backgrounds] et les comportements [Behaviors] ont des comportements. Par exemple, quand la surface de visualisation intersecte avec la limite d'application d'un arrière-plan particulier, cet arrière-plan est utilisé dans le rendu. Voir aussi *limites.*

limite d'influence (d'une lumière)
: Voir *influence (d'une lumière).*

limite de planification [scheduling bounds]
: Une région définie pour un comportement. Le volume d'activation d'une vue doit intersecter avec la surface de planification pour que le comportement devienne actif. Un comportement doit être capable de s'exécuter en réponse au stimulus. Voir aussi *actif, volume d'activation,* et *comportement.*

lumière ambiante
: Une source de lumière, qui fait briller dans toutes les directions, utilisée pour former les réflexions complexes entre les objets (réflexion répétée d'une lumière d'un objet vers un autre) existants dans le monde actuel. Voir également *couleur ambiante*.

limites
: Voir *volume de délimitation* et *classe Bounds*.

M

mode d'illumination
: Le calcul de la valeur d'illumination de chaque pixel pour l'illumination entre chaque sommet illuminé. Voir aussi *illumination, illumination Gouraud, illumination Flat.*

mode d'éclairage
: Le calcul de la couleur (illumination) d'un vertex spécifique d'un objet visuel résultant des influences des sources de lumières et des propriétés du matériau de l'objet visuel. L'illumination est le résultat des réflexions ambiantes, diffuses, et spéculaires comme de la couleur émise par le matériau. Voir aussi *réflexions ambiantes, réflexions diffuses, réflexions spéculaires* et *propriétés du matériau.*

monde virtuel
: L'espace conceptuel dans lequel les objets visuels « existent », étant définit par un seul objet Locale. Le monde virtuel est contenu dans un seul univers virtuel.

N

niveau de détail ou [LOD, level of détail]
: Le niveau de détail se réfère à l'application d'un comportement spécifique, qui provoque un changement dans la représentation d'un objet visuel, prenant comme base (normalement) la distance entre lui et le spectateur. Quand l'objet LOD est près du spectateur, une représentation plus détaillée est utilisée. La représentation perd du détail avec la distance. Le but étant de diminuer le temps de calcul du rendu sans dégrader le rendu de l'image. Voir *Partie 4.1* et *Chapitre 5* pour plus de détails.

normale
: Un vecteur qui défini l'orientation de la surface. En Java 3D, les normales sont associées aux points de coordonnées dans la géométrie. Voir aussi *Geometry.*

normale de la surface
: Voir *normale.*

O

objet
: Une instance de classe. Voir aussi *classe* et *objet visuel.*

objet cible
: Un objet du graphe scénique modifié par un comportement ou un déterminant.

objet de modification	Objet du graphe scénique qui est modifié par un comportement et par lequel le comportement affecte le monde virtuel. Par exemple, un objet TransformGroup est toujours l'objet de modification pour des comportements interactifs. Voir *Partie 4.2* pour plus d'information.
objet visuel	Le terme « d'objet visuel » est utilisé où le mot « objet » pourrait prendre place en Français (où Anglais) mais pas dans le sens de la Programmation Orientée Objet. Un objet visuel peut et peut ne pas être visible dans une vue, suivant plusieurs facteurs. Les « objets visuels » se réfèrent le plus souvent à un objet Shape3D (voir Partie 1.2). Voir aussi *volume*
œil local (éclairage)	Rendre une scène comme si elle était vue depuis la position locale de l'œil - à l'inverse de l'œil infini par défaut. Augmente le temps de rendu. L'œil infini est l'éclairage par défaut. Voir aussi *mode d'eclairage, réflexion spéculaire vecteur de l'œil* et *œil infini (éclairage).*

P

pixel	Un élément individuel de l'affichage. On peut donner une adresse a chaque pixel, par ses cordonnées [x, y] de position et lui assigner une couleur simple. En Java 3D, les programmes ne s'adressent pas individuellement aux pixels, en fait les éléments 3D sont tramés. Voir aussi *tramer.*
plan	Une surface plane étendue indéfiniment dans toutes les directions. L'orientation d'un plan est normalement exprimée par une normale de surface. Un plan peut etre défini par un seul point et un vecteur de normale.
pointage (picking)	La sélection d'un objet visuel pour une interaction avec la souris. Le pointage est exécuté en Java 3D par les comportements [behaviors]. Voir le Chapitre 4 pour plus d'informations sur les Behaviors (comportements), Picking (pointage), et les programmes d'exemple utilisant les classes de pointage. Voir aussi *comportement.*
pointer	Sélectionner un objet avec la souris. Voir aussi *pointage.*
police de caractère [font glyphe]	Une collection de caractères de l'alphabet. Un style de police possède ses propres tailles en points et attributs (i.e. *italique* ou **gras**). Voir aussi *caractères* et *style de police.*
polygone d'ombre	Un polygone utilisé pour créer une ombre dans une scène. Voir *Partie 6.7.3.*
polytope	Un volume de limitation par une intersection fermée de demi-espaces.
portée (d'une lumière)	La portion du graphe scénique pour laquelle l'influence d'une lumière est définie. Par défaut, la portée d'une lumière est définie par la branche du graphe dont elle est membre. Des sous-branches du graphe scénique peuvent êtres spécifiés comme la portée d'une lumière. Ce qui ne remplace pas les spécifications de la région d'influence d'une lumière, mais est complémentaire de l'influence d'une lumière. Voir aussi *influence (d'une lumière).*

ProcessStimulus	Une méthode de Behavior (comportement). La méthode processStimulus est appelée par java quand la condition de déclenchement appropriée à lieu. C'est par cette méthode que l'objet de comportement répond au déclencheur. Voir Chapitre 4 pour plus d'information. Voir également *comportement* et *condition de réveil.*
projecteur	La ligne qui lie les sommets d'un objet 3D dans l'espace des coordonnées de l'univers avec les pixels de la surface de l'image plate. Une ligne droite entre un sommet 3D et le point de vue (œil du spectateur) est un projecteur et determine le(s) pixel représentant le sommet devant êtres tramés
projection	Pour exprimer les cordonnées géométriques de l'univers géométrique des objets 3D dans l'espace de l'image plate.
propriétés du matériau	La spécification des couleurs ambiantes, diffuses, spéculaires et d'émission, et la convergence d'un matériau utilisé dans le calcul de l'illumination d'un objet. Les trois couleurs premières sont utilisées dans le calcul des réflexions adéquates. Voir aussi *réflexion ambiante, réflexion diffuse, réflexion spéculaire, mode d'eclairage,* et *illumination.*

Q

quaternion	Un quaternion est défini par quatre valeurs de points à virgule flottante |x y z w|. Un quaternion défini les rotations en trois dimensions.

R

rayon de pointage	Un rayon de pointage est un rayon dont le point extrême est la position de la souris et sa direction est parallèle à la projection (parallèle avec les projecteurs). Dans de nombreuses applications, le rayon de pointage est utilisé pour pointer un objet en vue d'une interaction. Aussi, PickRay est une sous-classe de PickShape. Voir le bloc de reference approprié ou la Documentation de l'API pour cette classe. Voir aussi *pointage.*
réflexion ambiante	La lumière produite par une source de lumière ambiante reflétée par un objet visuel ayant un matériau d'apparence appliqué. Le résultat dépend de la couleur ambiante de l'objet et de la couleur de la source de lumière ambiante. Voir *couleur ambiante, lumière ambiante* et *mode d'éclairage.*
réflexion diffuse	La plus commune des réflexions des objets visuels éclairés dans le monde réel. La couleur diffuse est la couleur d'un objet dans des conditions d'éclairage normales. Voir aussi *mode d'éclairage.*
réflexion spéculaire	Le reflet d'un éclairage depuis un objet brillant. Dans le mode réel la présence de la réflexion spéculaire dépend beaucoup de comment l'aspect de la surface. En Java 3D, ceci est paramètré par un matériau spéculaire et une valeur de concentration. Voir aussi *propriétés des matériaux, couleur spéculaire,* et *concentration.*

regard infini (éclairage) Regarder une scène comme si elle était vue depuis une position à l'infini. L'effet produit est d'avoir un vecteur de l'œil constant (0, 0, 1). Ce qui réduit le temps du rendu, mais peut avoir un effet «drôle » par le placement des réflections spéculaires. L'éclairage à l'infini de l'œil est la positon par défaut. Voir aussi *éclairage du modèle, réflexion spéculaire, vecteur de l'œil,* et *œil local (éclairage).*

région planifiée Voir *limite de planification.*

règle de la main droite 

Cette règle se réfère au lien entre la direction de vos doigts et de votre pouce de votre main droite. Ce procédé némotechnique est appliqué pour déterminer la face visible d'un polygone, quand on travaille avec un grand nombre de vecteurs, et que l'on veut savoir comment sont tournés les faces. La Figure sur la droite montre les doigts de la main droite tournant dans l'ordre dans lequel les coordonnées des sommets sont définies pour un triangle. Le pouce pointant vers le haut, indique que nous voyons la face avant de ce triangle. Voir aussi *face avant* et *cueillage*

moteur de rendu Logiciel pour produire l'image d'un graphe scénique.

rendre Produire une image représentée par le graphe scénique.

RVB [RGB] Rouge, Vert et Bleu, les trois composants utilises pour représenter la couleur.

RVBA Rouge, Vert, Bleu et Alpha, les trois composants utilises pour représenter la couleur avec une valeur de transparence.

S

set* ou méthode d'établissement La méthode d'une classe qui établit un champ de valeur dans un objet. Voir aussi *get**.

spectateur La (principale) personne qui visualise le dispositif visuel que Java 3D est censé rendre. C'est pour cet individu que les parametres de calibrations du PhysicalBody sont effectués.

style de police [typeface] Le style d'impression d'un texte. Par exemple Times Roman, Helvetica, et Courier sont des styles de police. Par comparaison, une police de caractère [font] est un style de police avec d'autres attributs spécifiques. Par exemple «10 points, italique, Times Roman» est une police de caractère. Voir aussi *police de caractère.*

T

traçage des rayons (ray tracing) Tâche qui rend les scènes par un mode de représentation individuel des rayons de lumière. Cette tache peut modéliser les effets internes aux objets comme les ombres mais n'est pas assez rapide pour le rendu en temps réel.

trame [raster] La mémoire par pixel du matériel d'affichage.

tramer [rasterize]	La conversion des objets visuels et leurs composants en leur image projetée. Le terme vient de l'usage de fait de matériel d'affichage par trame par tous les ordinateurs communs.
transformation	L'opération mathématique accomplie par un sommet, ou une collection de sommets, pour une translation, rotation, ou mise à l'échelle. Les transformations sont représentées par des matrices 4x4 et stockés dans les objets TransformGroup.
translation	Déplacer un sommet ou un ensemble de sommets.
triage de l'exécution [execution culling]	Comme les ressources de calcul sont partagées pour le rendu et l'exécution des comportements ; quand des comportements s'exécutent, plusieurs ressources de calcul (ou toutes) ne sont pas disponibles pour le rendu. Ce qui peut dégrader les performances à l'exécution des objets de comportement. Donc Java 3D ignores les comportements quand ils ne sont pas actifs pour préserver les ressources d'exécution. Ce qui est appelé «execution culling» (cueillage ou triage de l'exécution) puisque l'exécution des objets de comportements désactivés est cueillie et pour ainsi dire mise de coté. Voir aussi *actif* et *comportement.*
triangulation	La subdivision d'un polygone en triangles.
trois dimensions	Espace tridimensionnel.

U

univers virtuel	L'espace conceptuel dans lequel les objets visuels « existent ». Un univers [virtual universe] virtuel peut contenir de nombreux mondes virtuels qui seront définis par les objets Locale.

V

valeur alpha	Une valeur comprise entre 0.0 et 1.0, inclus. Les valeurs alpha sont utilisées de plusieurs manières. (e.g., valeur de transparence ou Interpolator (déterminant) pour les objets de comportement). Les variations temporelles des valeurs alpha sont produites par des objets Alpha. Voir aussi *classe Alpha, comportement,* et *déterminant.*
vecmath	Un package d'extension qui défini des classes pour le calcul de vecteurs mathématiques. Parmi elles les classes Tuple*.
vecteur d'une normale.	Voir *normale.*
vecteur lumière	Le vecteur entre la source de lumière et le vertex se trouvant illuminé. Voir aussi *mode d'éclairage.*
vecteur œil	Le vecteur entre un sommet (qui doit former une illumination) et la position de la visualisation. Ce vecteur est utilisé pour calculer la réflexion d'un objet. Voir aussi *réflexion spéculaire, œil local,* et *œil infini.*

vivant — Le terme de vivant se réfère au statut d'un objet du graphe scénique (sceneGraphObject) étant associé avec un moteur de rendu. En d'autres mots, un objet vivant est un membre d'une branche du graphe associé a une Locale (scène), laquelle étant membre du VirtualUniverse qui possède une View attaché (par l'intermédiaire de l'objet Locale) et, enfin, ayant la possibilité d'etre rendu. Les objets vivants ont des aptitudes réduites. Voir aussi *rendu, sceneGraphObject, branche du graphe, classe Locale, classe VirtualUniverse, graphe scénique*, et *aptitudes.*

volume [content] — Les objets visuels (incluant shapes et lumières) et sonores d'un univers virtuel. Voir aussi *objet visuel.*

volume d'activation — Un volume associé avec un Nœud ViewPlatform. Quand le volume d'activation entrecoupe une région d'application et une région planifiée, l'objet associé aux régions d'intersections, modifie le rendu. Les objets affectés comprennent les arrières plans, les comportements, et le brouillard. Voir également *ViewPlatform, limites d'application,* et *limites de planification.*

volume de délimitation [bounding volume] — Un volume qui définit la limite pour laquelle quelque chose est déterminée. Les limites sont utilisées avec des comportements, lumières, sons et autres dispositifs de Java 3D. Les limites sont définies dans un programme par les classes Bounds. Voir aussi c*lasse Bounds* et *région planifiée.*

W

X

Y

Yo-Yo — Un jouet.

Z

z-buffer — Une structure de données interne au rendu qui détermine la profondeur relative (distance depuis l'image plate) des objets visuels sur la base du pixel. Seuls les objets proches de l'image plate sont visibles.