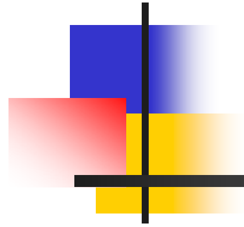


Flex Builder 3

Le 3D en Flex

http://www.flashsandy.org/tutorials/3.0/
http://www.petitpub.com/labs/media/flash/sandy3/

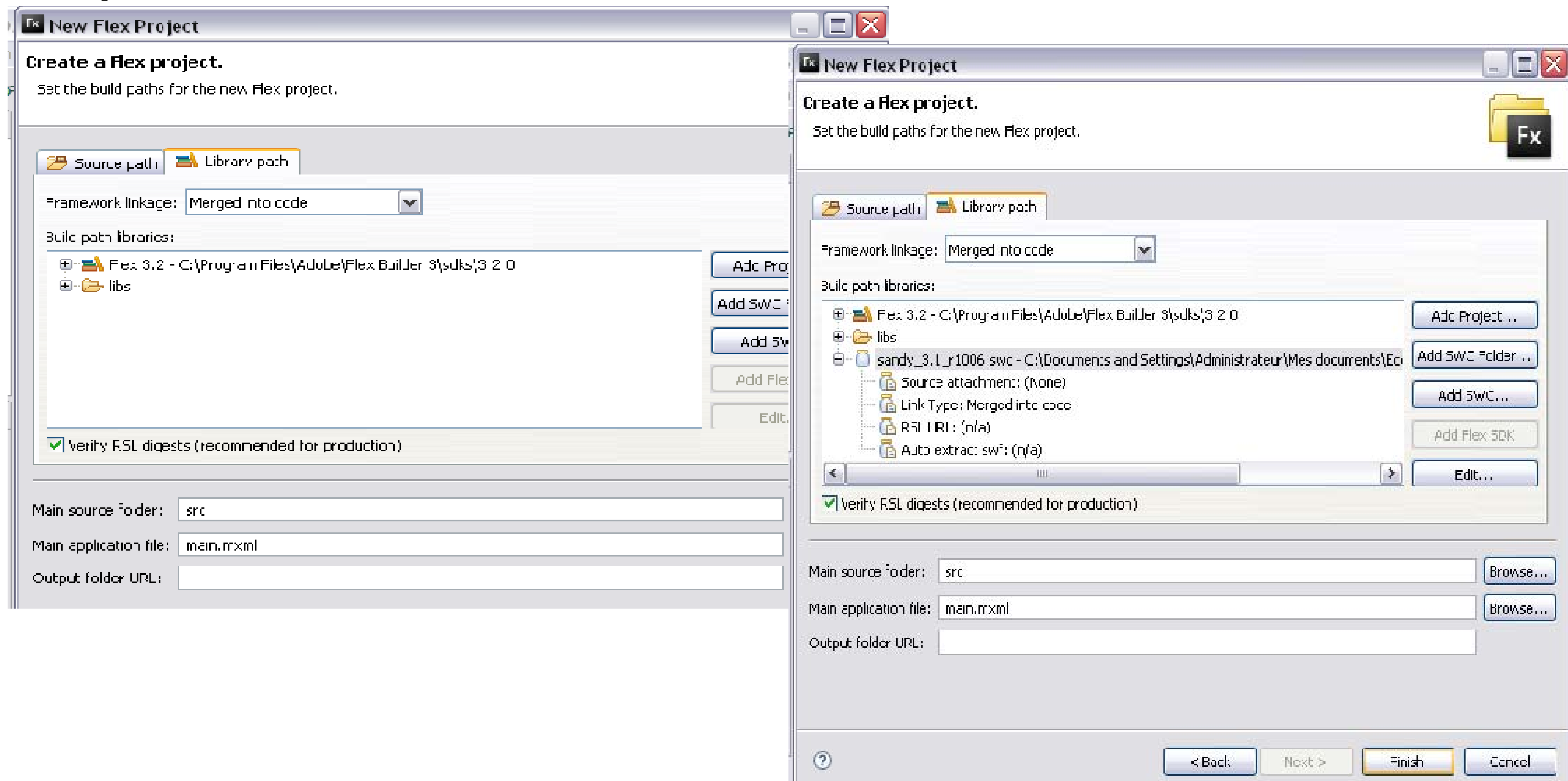
Préambule

- Flex ne permet pas de créer seul la 3D
  - Il passe par la librairie Sandy 3D
    - Créée pour Flash, elle s'adapte à Flex
    - Sandy 3.0.1 est la première version réalisée pour ActionScript3
    - Sources : www.flashSandy.org
  - Elle se charge dans Flex
    - à la création des projets Flex

Le logiciel Sandy

- Étapes de création
  1. Télécharger le fichier sandy_3.1_r1006.swc
  2. Après File>New project, choisir un nom puis cliquer sur Next
  3. Choisir le dossier de destination, puis refaire « Next »
  4. Une page s'ouvre, choisir l'onglet Library Path
  5. Dans cet écran, choisir « Add SWC ». Chercher le fichier SWC et l'ajouter au projet
  6. Terminer la création du projet par « Finish »

Le logiciel Sandy



Préambule

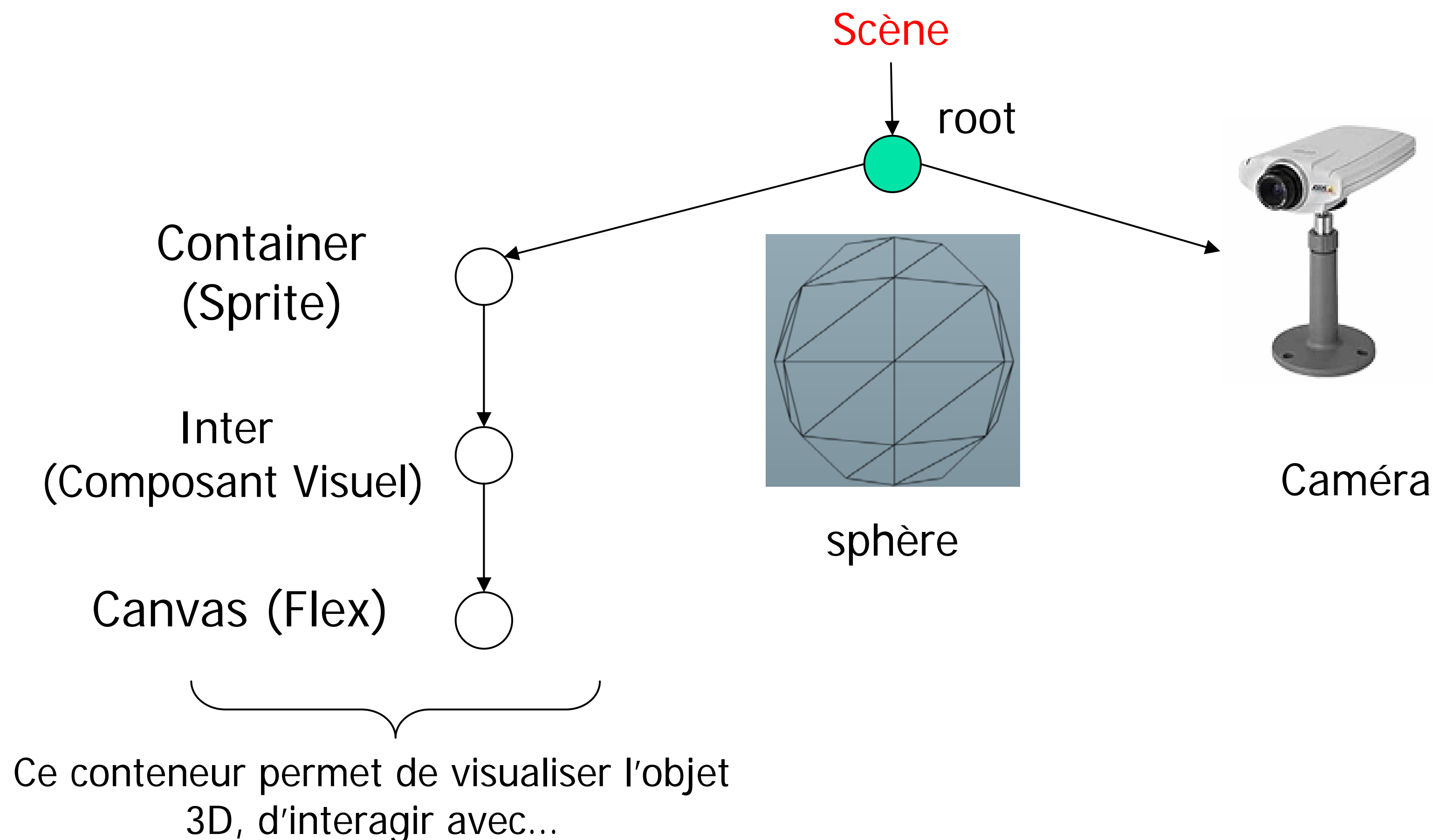
- Le monde de Sandy
  - Sandy affiche une représentation 2D avec des perspectives et des ombres donnant l'illusion d'une 3D
  - Le concept utilisé est proche du monde réel
    - On définit un espace 3D avec
      - des objets 3D dans différentes positions
      - une caméra (œil de l'observateur) regardant la scène à partir d'une certaine position
    - Les objets, ainsi que la caméra, peuvent être déplacés dans l'espace (le monde de Sandy), en créant une projection sur la rétine de l'œil

Préambule

- Le modèle de création 3D
  - C'est une structure géométrique (attachée à la forme 3D)
    - fondée sur la notion de polygones dans l'espace
  - Donne une apparence à la forme
    - En donnant une apparence à chaque polygone
      - Ainsi, chaque polygone peut avoir sa propre apparence
  - Une fois la scène créée, Sandy (le modèle) autorise les actions
    - Update : mise à jour par application de transformations locales
    - Rendering : transformation géométrique des objets visibles
    - Display : affichage des polygones avec les caractéristiques de couleur…

Pour créer la scène

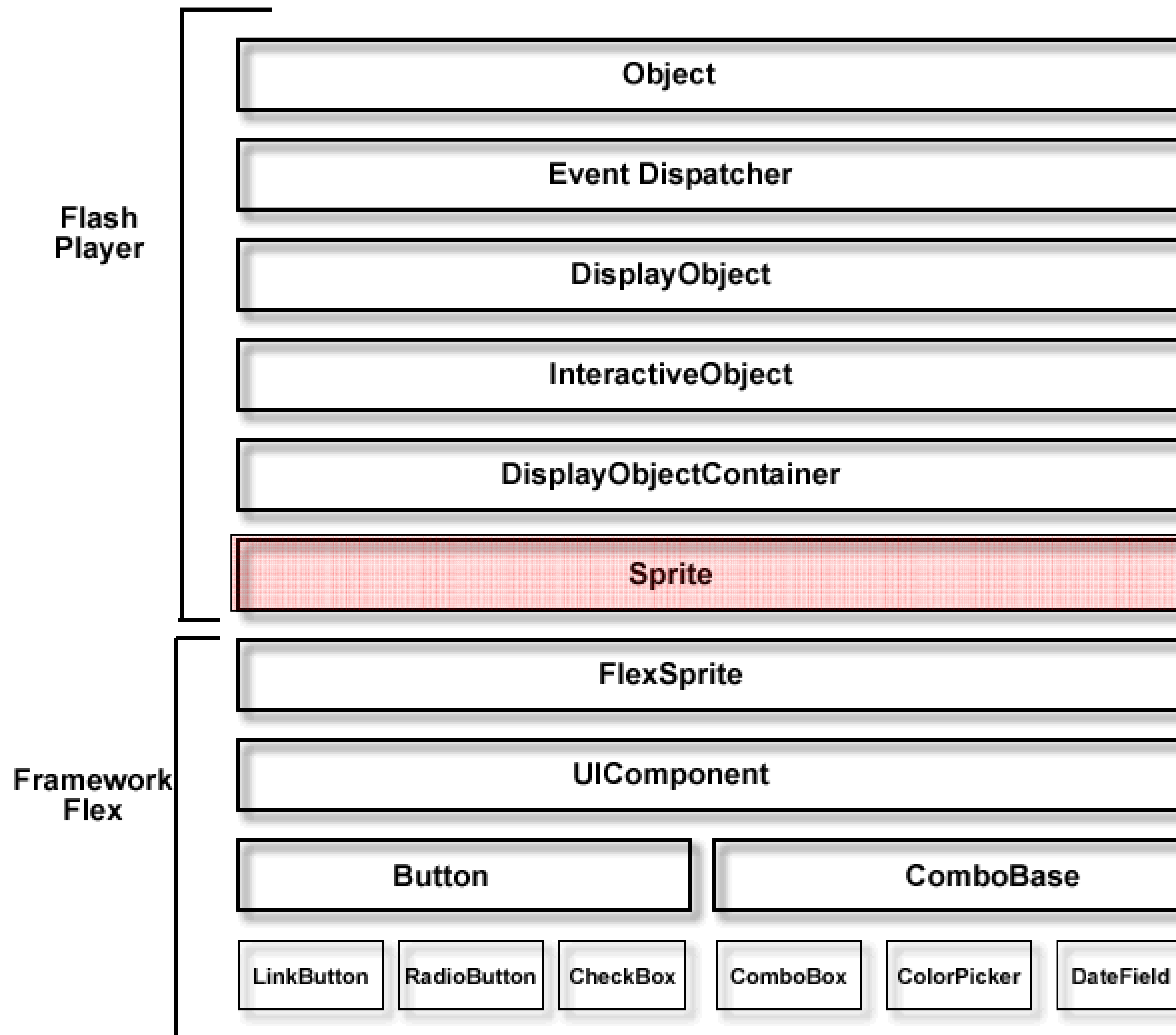
- La scène est formée d'un conteneur de type Flash, d'une caméra et d'un groupe graphique (root) auquel sera attaché l'objet graphique
- Le conteneur Flash permet de visualiser l'objet 3D suivant le point de vue de la caméra



Création de la scène

On utilise les composants visuels de Flex

- Hiérarchie des composants visuels
  - Les composants visuels Flex se comportent comme des objets d'affichage Flash
  - En effet, ceci se retrouve dans la chaîne d'héritage
    - Les composants visuels héritent de : mx.core.UIComponent qui hérite de mx.core.FlexSprite qui hérite directement de flash.display.Sprite qui fait partie de l'API Flash Player


Flash
Player
Object
Event Dispatcher
DisplayObject
InteractiveObject
DisplayObjectContainer
Sprite
Framework
Flex
FlexSprite
UIComponent
Button
ComboBase
LinkButton
RadioButton
CheckBox
ComboBox
ColorPicker
DateField

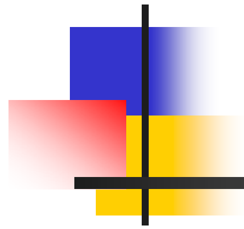
Création de la scène

- Prenons un exemple
  - Construire un objet sphère
  - L’exemple suivant montre comment la scène est construite par
    - création du conteneur (Sprite)
    - création du composant visuel et son ajout au conteneur
    - création de la scène par association de la root, de la caméra et du conteneur
    - ajout de l’objet sphère à la scène
    - Définition d’un rendu à la scène

Prenons un exemple : sphère : Simple_ex.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
    creationComplete="init()" width="518">
<mx:Script>
<![CDATA[
    import mx.core.UIComponent;
    import sandy.core.Scene3D;
    import sandy.core.scenegraph.*;
    import sandy.primitive.*;

    private var scene:Scene3D;

    private function init():void{
        //Définition d'un composant visuel Flex
        var inter:UIComponent=new UIComponent;

        //Création d'un conteneur Sprite pour y mettre une forme
        var container:Sprite = new Sprite;

        //Ajout du composant visuel à ce container (pour afficher la scène)
        inter.addChild(container);
        MainCanvas.addChild(inter);
```

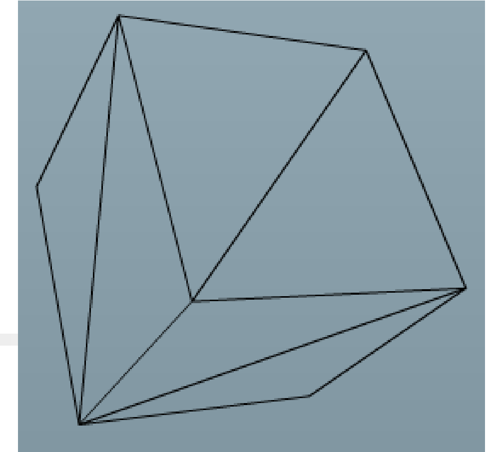
```
    /* création de la scène. On lui donne ce conteneur, une caméra (un
    point de vue), et un groupe root où ajouter les objets*/
    scene=new Scene3D("maScene", container, new
    Camera3D(500,400), new Group("root"));

    //La caméra est placée au point : (500, 400) avec un recul de 0 par
    rapport à la scène

    // On ajoute un objet à la scène que l'on vient de créer
     scene.root.addChild(new Sphere("maScene"));

    //On donne un rendu à la scène
    scene.render();
    }
]]>
</mx:Script>

//Déclaration du conteneur de la scène
<mx:Canvas id="MainCanvas" width="100%" height="100%">
</mx:Canvas>
</mx:Application>
```



Flex 3D

- Commentaires
  - Les imports
    - Il faut importer les classes de Sandy même si la librairie est ajoutée au projet
  - Les variables importantes : 2 concepts forts de Sandy
    - Scene3D : conteneur principal des objets
    - Camera3D : point de vue de l'utilisateur des objets initiaux

Flex 3D



- Deuxième exemple
  - Création d’un cube : ex001.mxml
  - On retrouve la même stratégie de construction
  - On reprécise certains éléments, comme :
    - initialisation de la caméra:

      camera = **new** Camera3D(300, 300);

    ➔ position par défaut : (0,0,0)

      camera.z = -400;

    ➔ Régler la caméra à -400 = reculer de 400px de l'origine de la Scène
    - *Ajout d'un EventListener*

      addEventListener(Event.ENTER_FRAME, enterFrameHandler);

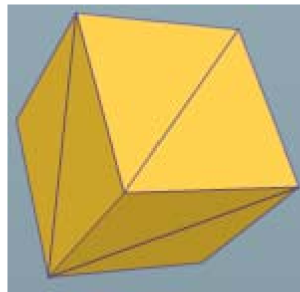
    - *qui dit à la scène de s’initialiser*
  - *On tourne le cube pour donner l’illusion de la 3D*

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute" creationComplete="init()">
<mx:Script><![CDATA[
    import mx.core.UIComponent;import flash.display.Sprite;
    import flash.events.*;import sandy.core.Scene3D;
    import sandy.core.data.*;import sandy.core.scenegraph.*;
    import sandy.materials.*;import sandy.materials.attributes.*;
    import sandy.primitive.*;

     private var scene:Scene3D; private var camera:Camera3D;

    private function init():void{
     {
    // Crée la caméra
     camera = new Camera3D( 300, 300 );
     camera.z = -400;
     // Crée la racine (root) de l'arbre qui va représenter la scène du cube
      var root:Group = createScene();
     // Associe dans l'arbre : la root, la caméra et le composent visuel
    scene=new Scene3D("maScene", container, new Camera3D(500,400),
    root);
    // Ajoute un écouteur à la scène
      addEventListener( Event.ENTER_FRAME, enterFrameHandler );
   }
```

```
    // Crée le graphisme de la scène et l'attache à un groupe
     function createScene():Group
    { // Crée une root pour la scène
       var g:Group = new Group();

       // Crée un cube
       var box = new Box("box",100,100,100);
        box.rotateX = 30;box.rotateY = 30; // donne l'illusion 3D

      // Ajoute le cube au Groupe
     g.addChild(box);
     return g;
    }
    // L' "Event.ENTER_FRAME" event handler dit à la scène de s'initialiser
    function enterFrameHandler( event : Event ) : void
    {
       scene.render();
     }
}
]]>
</mx:Script>
<mx:UIComponent id="container" width="100%" height="100%"/>
</mx:Application>
```

Flex 3D

- Notion de Skin (habillage)
  - Dans Sandy3D, si nous ne définissons pas d'apparence (skin), les objets sont rendus sous forme de fil de fer
  - Autrement, le skin : va d'une simple couleur à un support vidéo
  - Pour introduire le skin, on utilise le couple d'attributs :
    - Matière (Material) / apparence (Appearence)
  - Dans la version 3 de Sandy :
    - 9 types de matières
  - Exemple : habiller le cube précédent : ex002.mxml



Flex 3D

Notion de Skin

- Les étapes :
  1. Création de l'attribut matière

```
var materialAttr:MaterialAttributes =
new MaterialAttributes(new LineAttributes( 0.5,
    0x2111BB, 0.4 ), new LightAttributes( true, 0.1))
```

- **LineAttributes** : précise les *dimensions*, *couleur* et *transparence* des lignes utilisées pour dessiner le cube
- **LightAttributes** : précise la présence ou non de la lumière et le niveau de lumière

– Ces attributs doivent implémenter l'interface IAttributes

Flex 3D

Notion de Skin

http://wiki.mediabox.fr/tutoriaux/flex/sandy3d

2. Application de l'attribut material
   - Dans cet exemple, nous l'appliquerons à ColorMaterial qui est le matériau le plus facile à utiliser

```
var material:Material = new ColorMaterial(
    0xFFCC33, 1, materialAttr );
```

   - Le ColorMaterial prend trois paramètres :
     - la couleur des faces du cube
     - la transparence de la couleur
     - et l'attribut du matériel que nous venons de définir

Flex 3D

Notion de Skin

http://wiki.mediabox.fr/tutoriaux/flex/sandy3d

3. Réglage de la lumière pour la matière
   - On demande au moteur de rendu d'utiliser la lumière

     ```
     material.lightingEnable = true;
     ```

   ➔ Les couleurs des faces seront initialisées avec des ombres :
     - ➔ face frontale plus lumineuse, tandis que les faces de derrière moins
4. Application (affectation) de l'apparence à la matière

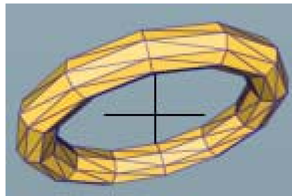
   ```
   var app:Appearance = new Appearance( material );
   ```

5. Affectation de l'*apparence* à l'objet concerné, ici le cube

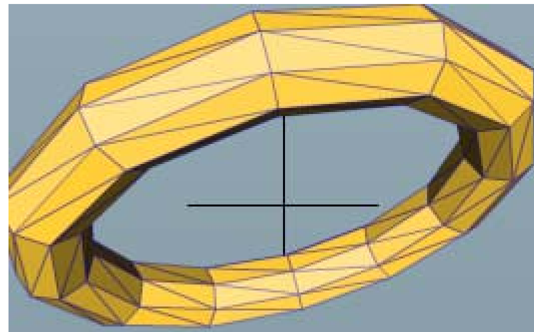
   ```
   box.appearance = app;
   ```

Comprendre la notion de caméra

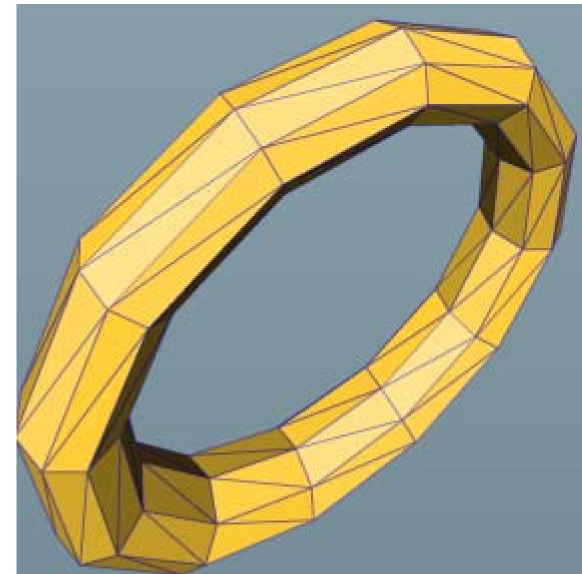
- Notion de point de vue de la scène
  - Maintenant, on cherche à modifier la position de la caméra pour changer de point de vue
  - Exemple : ex003.mxml



Éloignement



Rapprochement



Positionnement de la caméra à droite

Comprendre la notion de caméra

- Camera3D
  - **C’est plus qu'un objet statique** : on peut la bouger autour d'un objet, la tourner...
- Problème
  - Est-il préférable de bouger les objets autour de la caméra, ou à l'inverse, bouger la caméra autour des objets?
- Le meilleur conseil est :
  - Si on veut regarder bouger un seul objet devant la caméra, en laissant les autres dans leur position originale, il vaut mieux bouger l'objet seulement
  - Si on veut bouger tous les objets autour de la caméra, il vaut mieux bouger uniquement la caméra, et non les objets

Comprendre la notion de caméra

- Retour sur l'exercice : ex003.mxml
  - On définit la position de la caméra par :

    camera = new Camera3D(300, 300);

  - On peut redéfinir ensuite sa position

    camera.x = 100; // bouge la caméra vers la droite

    camera.y = 100; // bouge la caméra vers le haut

  - On peut dire à la caméra là où regarder dans la scène

    camera.lookAt(0,0,0); // définit la direction vers laquelle regarder

Comprendre la notion de caméra

- *La création de la scène 3D dans* la fonction createScene(),
  - Débute par la création d'un système de coordonnées en dessinant 3 lignes
  - Crée ensuite l'objet tore

    var torus:Torus = new Torus("theTorus", 120, 20);

    - *3 paramètres : le nom, le rayon de l'anneau et le rayon de l'anneau central*
  - Définit la matière et l'apparence
  - Rattache le système de coordonnées et l'objet tore au groupe
- La gestion de l'événement
  - Consiste à déplacer, tourner le tore en utilisant les touches clavier

Comprendre la notion de caméra

– *On ajoute des écouteurs d'événements pour faire des déplacements de caméra et observer suivant différents points de vue le Tore*

```
addEventListener0; Event.ENTER_FRAME,
  enterFrameHandler );
stage.addEventListener0;
  KeyboardEvent.KEY_DOWN, keyPressed);
```

- Le premier écouteur est responsable du rendu de toute la scène
- tandis que l'autre intercepte les événements saisis par l'utilisateur : ici le clavier

- la méthode keyPressed:

```
switch(event.keyCode) {
    case Keyboard.UP: //rotation dans un plan vertical
      camera.tilt +=2; break;
    case Keyboard.DOWN:
      camera.tilt -=2; break;
    case Keyboard.RIGHT:
      camera.pan -=2;  break; //rotation dans un plan horizontal
    case Keyboard.LEFT:
      camera.pan +=2; break;
    case Keyboard.CONTROL:
      camera.roll +=2; break;
    case Keyboard.PAGE_DOWN:
      camera.z -=5; break; //éloignement
    case Keyboard.PAGE_UP:
      camera.z +=5; break; //rapprochement
  }
```

- Utiliser les touches clavier pour déplacer le Tore
- Vous pouvez remarquer les changements de points de vus qui sont opérés

Comprendre la notion de caméra

- Exercice
  - On bouge la caméra et non l'objet; l'objet est stationnaire
  - Pour s'en rendre compte, on peut modifier l'exemple ex003.mxml de façon à ce que :
    - les positions camera.x et camera.y soient 0 (le défaut)
  - Agissez maintenant sur l'axe Z
    - En laissant camera.x et camera.y réglés à 100
    - Cela veut dire que la caméra regarde « tout droit »
  - Enfin, la caméra regarde le point (0,0,0) qui est le centre des axes
  - Jouez avec les exemples pour se fixer une idée de comment gérer la caméra

Comprendre la notion de caméra

- Bouger maintenant les objets autour de la caméra
  - Dans la partie précédente, nous avons montré comment faire bouger la caméra autour des objets donnant l'effet que les objets bougent autour d'elle
  - Mais ils étaient en fait tous immobiles à leur place
  - Nous voulons maintenant les faire bouger !
    - On peut déplacer un objet sur l'écran, en déplaçant un objet unique ou en bougeant un groupe contenant un ou plusieurs objets…

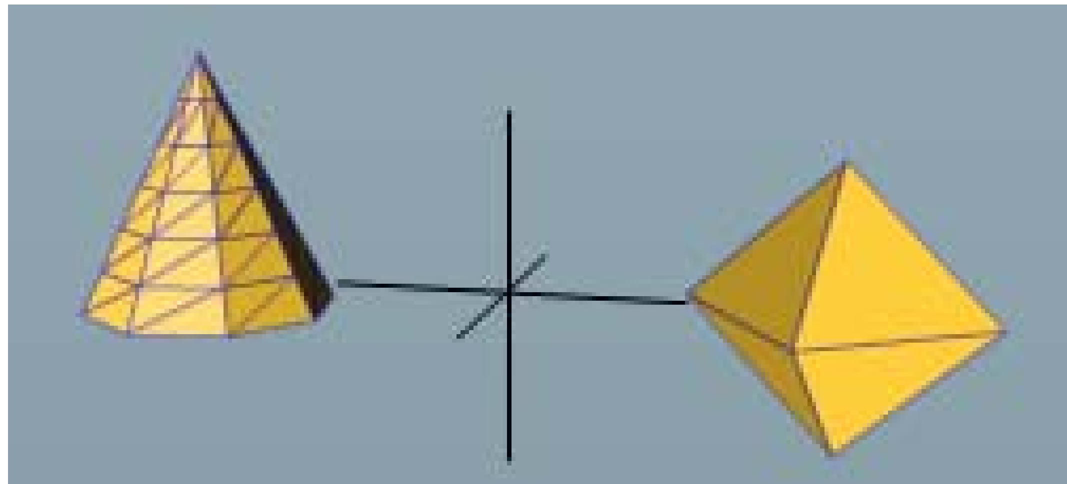
Bouger des objets autour de la caméra

- Principe
  - Pour déplacer un seul objet,
    - Une fois que vous avez sa position, vous pouvez le faire pivoter autour d’un des trois axes du système de coordonnées en utilisant les propriétés
      - rotateX, rotateY, et rotateZ
    - Ensuite, utilisez les propriétés
      - pan et tilt

      pour faire pivoter l'objet autour de ses axes locaux
    - Si vous souhaitez déplacer un groupe d'objets, vous avez besoin d'introduire une nouvelle classe, nommée
      - TransformGroup

Bouger des objets autour de la caméra

- Exemple d’illustration : ex004.mxml
  - Ici, nous allons dessiner 2 objets et montrer comment bouger un objet et bouger un group d’objets



Bouger des objets autour de la caméra

- Comment faire ? Examinons le code
  1. *Déclaration des objets nécessaires*
     - D'abord, on doit définir les variables objets en privé

```
// le groupe transformable
private var tg:TransformGroup;
// le cône
private var myCone:Cone;
// le tétraèdre
private var myHedra:Hedra;
```

     - Cela nous permet de nous référer à ces variables depuis partout dans la classe

Bouger des objets autour de la caméra

2. *Création du TransformGroup*
   - Dans la méthode createScene(), on crée l'objet tg par instanciation de la classe TransformGroup

     tg = new TransformGroup('myGroup');

     - TransformGroup apportera deux nouveaux éléments : un Cône et un tétradèdre
3. Création et positionnement des objets
   - Nous créons ensuite les deux nouveaux objets et on les positionne correctement pour qu'ils ne se chevauchent pas

```
myCone = new Cone("theObj1",50, 100);
myHedra = new Hedra( "theObj2", 80, 60, 60 );
myCone.x = -160;
myCone.z = 150;
myHedra.x = 90;
```

Bouger des objets autour de la caméra

4. On peut ajouter une apparence pour chaque objet

```
myCone.appearance = app;
myHedra.appearance = app;
```

5. On les ajoute à la transformGroup
   – Ainsi, nous aurons une référence à un «nœud» que l'on peut déplacer comme on veut

```
tg.addChild(myCone);
tg.addChild(myHedra);
```

6. On ajoute les objets restant au groupe principal

```
g.addChild(tg);
g.addChild(myXLine);
g.addChild(myYLine);
g.addChild(myZLine);
```

Bouger des objets autour de la caméra

- Remarque
  - Attention, on doit juste ajouter :
    - ❖ le TransformGroup que l'on veint de créer et les trois lignes
    - ❖ On n'ajoute pas les deux autres objets puisqu’on les a déjà ajoutés au TransformGroup

Bouger des objets autour de la caméra

7. Gérer les événements
   - Voyons voir quels événements nous gérons ?
   - Dans la fonction enterFrameHandler, on a décidé de montrer comment vous pouvez accéder aux objets simples, directement
   - Ainsi, nous pouvons faire pivoter chaque objet, à titre individuel

```
myHedra.pan +=4;
myCone.pan +=4;
scene.render();
```

Bouger des objets autour de la caméra

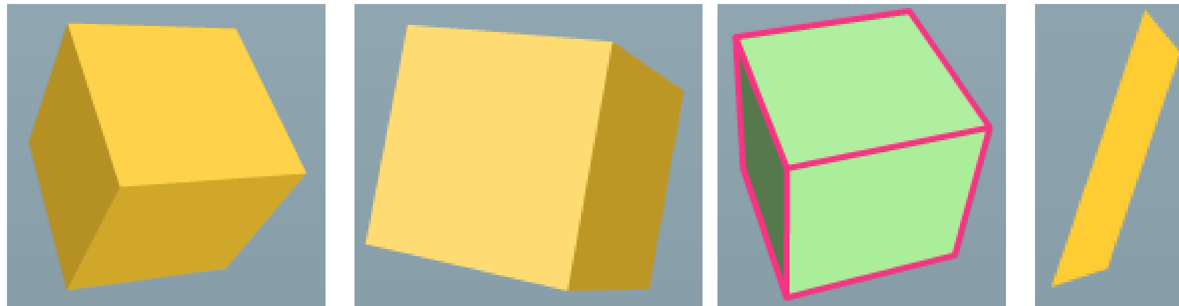
7. Gérer les événements
   - Dans la fonction keyPressed, on a voulu montrer comment accéder à l'ensemble du Groupe
   - Ainsi, vous pourrez voir le mouvement de chacun des deux objets ensemble

```
switch(event.keyCode) {
    case Keyboard.UP: tg.y +=2; break;
    case Keyboard.DOWN: tg.y -=2; break;
    case Keyboard.RIGHT: tg.roll +=2; break;
    case Keyboard.LEFT: tg.roll -=2; break;
}
```

8. Et maintenant tester :
   - Appuyez sur les flèches du clavier pour faire tourner l'ensemble du groupe
   - Notez que les trois axes ne se déplacent pas, puisque nous ne vous déplacez pas la caméra. Et les axes ne sont pas à l'intérieur du TransformGroup

Bouger des objets autour de la caméra

- Bouger l’objet par rapport à la caméra et agir sur son apparence : skin_08.mxml



  - Nous avons déjà vu précédemment comment appliquer une matière simple à un cube
  - Nous allons commencer à partir de là et nous passerons à des effets plus sophistiqués, en agissant sur les polygones du cube

Bouger des objets autour de la caméra

- Le code
  - Vous avez dû remarqué qu'on a utilisé trois variables d'apparence
    - private var app01:Appearance;
    - private var app02:Appearance;
    - private var app03:Appearance;

    car nous voulons utiliser les touches clavier pour les changer de manière interactive
  - Pour chaque attribut, appi, on utilise une matière particulière, ex pour la première :

```
var material01:Material = new ColorMaterial( 0xFFCC33, 1,
    materialAttr01 );
material01.lightingEnable = true;
```

Bouger des objets autour de la caméra

- Avec le deuxième type d’apparence, on définit en plus des lightAttributes, des LineAttribues
- var materialAttr02:MaterialAttributes = new MaterialAttributes(new LightAttributes(true, 0.1), new LineAttributes(3, 0xF43582, 1));
- Les LineAttribues montrent comment sont rendues les lignes qui composent notre modèle
- Le nombre de lignes qui construisent nos modèles repose sur le mode que vous avez choisi, ici 4 car le mode est quad comme quadrilatère
- LineAttribues a 3 paramètres:
  - l'épaisseur de la ligne
  - sa couleur
  - et la valeur alpha en % de l'opacité complète

Bouger des objets autour de la caméra

- Enfin, avec la 3ème apparence, on définit en plus ce qu'on appelle : OutlineAttributes

```
var materialAttr03:MaterialAttributes = new
MaterialAttributes( new LightAttributes( true, 0.1), new
OutlineAttributes(3, 0xFC5858, 1), new LineAttributes(1,
0x000000, 1) );
```

- Ceci permet de définir l'apparence du périmètre de l'objet : c'est la ligne qui entoure l'objet
- On peut également donner une apparence particulière à chaque face du cube

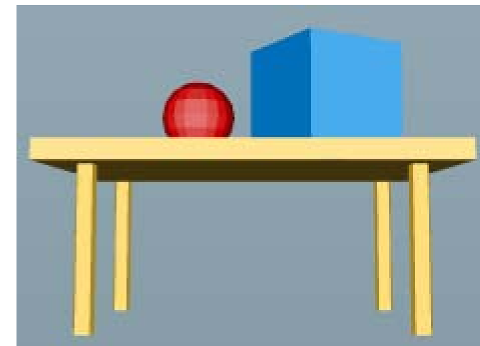
```
box.aPolygons[0].appearance = app01;
box.aPolygons[1].appearance = app02;
box.aPolygons[2].appearance = app03;
box.aPolygons[3].appearance = app02;
box.aPolygons[4].appearance = app01;
box.aPolygons[5].appearance = app03;
```

Bouger des objets autour de la caméra

- Exercice
  - Maintenant, vous devriez être en mesure de combiner les notions apprises dans cette section et la section précédente, afin de déplacer l'objet, et afin de déplacer des objets fixes

Maîtriser la lumière dans Sandy 3D

- Introduction
  - Quand on veut afficher des objets, on veut aussi utiliser les effets de lumière qui sont possibles initialement pour chaque objet
  - On peut être satisfait par les réglages par défaut, mais on peut vouloir être capable de modifier l'intensité de la lumière, ou bien sa direction
- Illustration : ex0041.mxml
  - Examinons le code



Maîtriser la lumière dans Sandy 3D

Les étapes

1. **Variables X, Y, Z**
   - Pour contrôler la direction de la lumière, puisque nous sommes dans un monde en 3D, nous avons besoin de 3 variables, X, Y et Z, que nous définissons comme variables d'instance
   - Ce faisant, nous serons capables de contrôler leur valeur en utilisant les touches clavier

```
private var lightX = 0;
private var lightY = 0;
private var lightZ = 10;
```

Maîtriser la lumière dans Sandy 3D

2. Régler la direction de l’éclairage
   - Syntaxe

     scene.light.setDirection(lightX, lightY, lightZ);
   - Comme vous pouvez le voir, il vous suffit d'obtenir la propriété lumière qui appartient à l'objet Scene3D et définir la valeur que vous avez décidé de créer
3. Ensuite, dans la fonction createScene ()
   - On a mis beaucoup d'objets pour simuler une table avec une balle avec une boule et un cube
   - La table est une boîte à quatre cylindres et 4 pieds et la balle est notre sphère
   - La plateau de la table et la sphère ont été construits à partir de primitives simples de Sandy
4. Le reste est facile à comprendre…

Maîtriser la lumière dans Sandy 3D

5. Création du gestionnaire d'entrée
   - Maintenant, voyons ce qu'il y a dans le handler keyPressed(event:KeyboardEvent)
     - case Keyboard.PAGE_DOWN: scene.light.setPower0;scene.light.getPower 0;) - 5); break;
     - case Keyboard.PAGE_UP: scene.light.setPower0;scene.light.getPower 0;) + 5); break;
     - case Keyboard.UP: lightY+=10; scene.light.setDirection0;lightX, lightY, lightZ); break;
     - case Keyboard.DOWN: lightY-=10; scene.light.setDirection0;lightX, lightY, lightZ); break;
     - case Keyboard.RIGHT: lightX+=10; scene.light.setDirection0;lightX, lightY, lightZ); break;
     - case Keyboard.LEFT: lightX-=10; scene.light.setDirection0;lightX, lightY, lightZ); break;

Maîtriser la lumière dans Sandy 3D

5. Création du gestionnaire d'entrée
   – avec page UP, page DOWN key, on contrôle l'intensité de la lumière,
   – Tandis que avec l'arrow key, vous contrôlez la direction de la lumière. Simple!
6. Essaye le
   – Si ça ne marche pas, essayez ici : http://www.flashsandy.org/tutorials/3.0/sandy_cs3_tut04 1

Introduction à Sprite 3D

- Idée
  - Nous avions travaillé précédemment avec Sprite2 où on choisissait la face de rotation de l’objet
  - Dans Sprite3D, nous n'avons pas seulement une face de l'objet à rendre graphiquement, mais il est possible d'afficher 360 faces de chaque objet, et ce sera la rotation de la caméra, ou bien de l'objet, qui déterminera la bonne face à présenter à l'utilisateur
  - Pour travailler avec Sripte 2D, une seule image était suffisante
  - Pour travailler avec Sprite 3D, on doit disposer d’un fichier SWF qui est une séquence d’images pré-rendues
  - Cette section va essentiellement se concentrer sur la façon de préparer votre objet Sprite3D et comment l’utiliser

- Démarrage
  - Pour travailler avec Sprite3D nous avons besoin de créer deux fichiers swf différents
  - Le premier sera utilisé pour préparer l'objet Sprite3D, qui n'a rien à voir avec Sandy
  - Le second est notre programme qui utilise la bibliothèque Sandy qui utilisera le fichier swf préparé
  - Donc, démarrons avec ce fichier

Introduction à Sprite 3D

- L’objet Sprite 3D
  - L'objet Sprite3D n'est rien d'autre qu'un fichier swf simple, avec 360 images
  - Donc ce que nous devons faire en premier est de choisir un objet qu’on tranche en 360 images (au maximum), afin d'avoir les primitives pour créer notre fichier SWF
  - Le répertoire plane contient les 360 images obtenues par un créateur de gif animé

Introduction à Sprite 3D

- Voici un exemple d'un gif animé qui permet de voir ces 360 images

- Les étapes

1. Chargement du swf
   - on utilise une fonction pratique de sandy : Load

```
queue = new LoaderQueue();
queue.add( "plane", new
    URLRequest("plane/plane.swf") );
queue.addEventListener(SandyEvent.QUEUE_COM
    PLETE, loadComplete );
queue.start();
```

2. Pour utiliser l'élément importé, juste une ligne de code :

```
s = new Sprite3D("plane",queue.data["plane"],2);
```

   - Le nombre « 2 » est utilisé pour doubler la taille du fichier swf
   - On peut changer ce paramètre du constructeur pour redimensionner des images facilement

Introduction à Sprite 3D

3. *Gestion de l'objet créé avec le swf*

- La fonction keyPressedHandler permet de changer la direction de l'objet avec les touches DROITE et GAUCHE

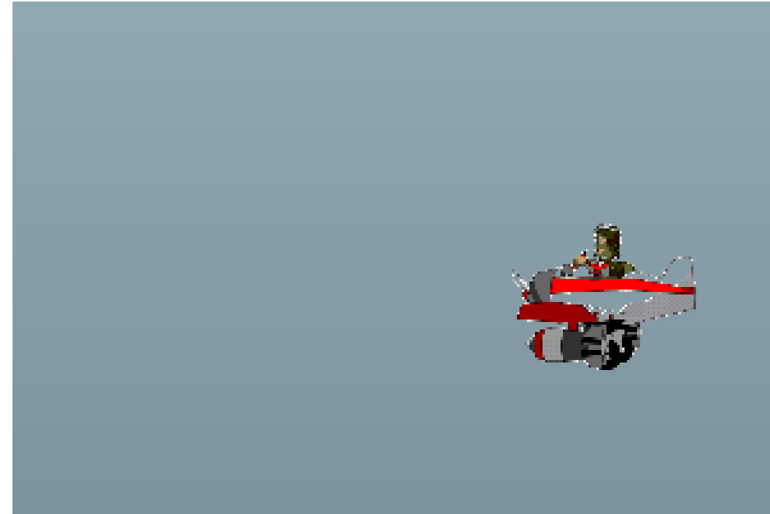
```
private function
    keyPressedHandler(event:KeyboardEvent):void {
     if(event.keyCode == Keyboard.RIGHT)
           s.rotateY -=5;
     if(event.keyCode == Keyboard.LEFT)
           s.rotateY +=5;
     }
```

Introduction à Sprite 3D

3. *Gestion de l'objet créé avec le swf (suite)*
   - On a ensuite besoin d'une fonction qui empêche l'objet de disparaître, qu'on a placé dans l'enterFrameHandler
   - L'objet réapparaîtra sur la gauche s'il va trop loin

```
private function enterFrameHandler( event : Event ) :
    void
    {  if (s.rotateY==0) s.rotateY=0.1;
           if(s.x > 220 && s.z < 0)  s.x=-220;
           else if(s.x < -220 && s.z < 0) s.x=220;
           else if (s.z<-250) s.z=1000;
          s.moveForward(-7);
          scene.render();
    }
```

Introduction à Sprite 3D



Importation de modèles

- Objectif :
  - Importer des modèles venant d’autres fichiers
  - Sandy peut importer des fichiers au format 3DS MAX, et Collada (format non standard)
  - En réalité, Il y a deux façons pour utiliser un modèle 3D provenant d'un autre logiciel :
    - utiliser un modèle qui a été exporté dans un format AS
    - importer un fichier 3DS : vous verrez qu'il faudra un peu plus de code à écrire

Importation de modèles

- Méthode 1 : importing3DS.mxml
  - Le modèle est rangé dans Tereira.as. Vous pouvez jeter un coup d'œil dessus
  - Dans l'application Flex, on crée une variable pot en instanciant la classe Teiera
    - private var scene:Scene3D;
    - private var camera:Camera3D;
    - private var pot:Teiera;
    - pot = new Teiera("pot");
  - Après, on l'utilise comme tout autre objet

Importation de modèles

- Méthode 2 : importing3DS_1.mxml
  - Le modèle est rangé dans teieraASE.ASE
  - La principale différence avec ce qui précède, est qu'il faut importer un parser de Sandy

    import sandy.parser.*;
  - Après, on définit un objet de type Shape3D pour pouvoir interagir avec
    - private var pot:Shape3D;
  - Ensuite vient la partie importante qui importe le fichier 3DS. On essaie de prévenir les erreurs qui peuvent provenir lors de l'import
    - var parser:IParser = Parser.create("asset/teieraASE.ASE",Parser.ASE);
    - parser.addEventListener(ParserEvent.FAIL, onError);
    - parser.addEventListener(ParserEvent.INIT, createScene);
    - parser.parse();

Importation de modèles

- Un autre exemple d'import d'un format 3DMAX : importing3DS_2.mxml
  - Ici, nous apprenons à utiliser la classe **ParserStack** qui ressemble à la classe déjà vue **LoaderQueue**
  - La classe LoaderQueue permet de charger certains visuel assets (images, swf files) tandis que ParserStack nous permet de charger différents modèles 3D basés sur le parsing de fichiers 3D

Importation de modèles

- Un autre exemple d'import d'un format 3DMAX : importing3DS_2.mxml
  - Tout d'abord nous avons besoin de définir les éléments IParser, un pour chaque modèle que nous voulons charger : 4 roues et 1 chassis

```
var parser:IParser =
Parser.create("assets/models/ASE/car.ASE",Parser.ASE ); var
parserLF:IParser =
Parser.create("assets/models/ASE/wheel_Front_L.ASE",Parse
r.ASE ); var parserRF:IParser =
Parser.create("assets/models/ASE/wheel_Front_R.ASE",Pars
er.ASE ); var parserLR:IParser =
Parser.create("assets/models/ASE/wheel_Rear_L.ASE",Parse
r.ASE ); var parserRR:IParser =
Parser.create("assets/models/ASE/wheel_Rear_R.ASE",Parse
r.ASE );
```

Importation de modèles

- Un autre exemple d'import d'un format 3DMAX : importing3DS_2.mxml
  - Une fois que nous avons créé ces IParser, nous les ajouterons à notre classe PareserStack:
    - parserStack.add("carParser",parser);
    - parserStack.add("wheelLFParser",parserLF);
    - parserStack.add("wheelRFParser",parserRF);
    - parserStack.add("wheelLRParser",parserLR);
    - parserStack.add("wheelRRParser",parserRR);

Importation de modèles

- Dans la méthode parserComplete (..), on extrait les objets Shape3D que nous venons de charger
- Pour ce faire nous allons utiliser une méthode publique fournie par la classe ParserStack: (..): getGroupByName
  - car =parserStack.getGroupByName("carParser").children[0] as Shape3D;
  - wheelLF = parserStack.getGroupByName("wheelLFParser").children[0] as Shape3D;
  - wheelRF =
  - parserStack.getGroupByName("wheelRFParser").children[0] as Shape3D;
  - wheelLR =
  - parserStack.getGroupByName("wheelLRParser").children[0] as Shape3D;
  - wheelRR =
  - parserStack.getGroupByName("wheelRRParser").children[0] as Shape3D;

Importation de modèles

- Maintenant que tous les objets dont nous avons besoin sont créés nous avons besoin de charger les skins pour nos Camero
- Pour ce faire nous allons utiliser la classe LoaderQueue qui se charge d'affecter deux peaux différentes : une pour les roues et un pour le châssis
- var material:BitmapMaterial = new BitmapMaterial(queue.data["carSkin"].bitmapData); … var materialW:BitmapMaterial = new BitmapMaterial(queue.data["wheels"].bitmapData);
- Ces deux materiaux seront utilisés comme entrées pour l'apparence du chassis et des roues pour notre caméra

```
tg = new TransformGroup('myGroup');
…
tg.addChild( wheelLF );
tg.addChild( wheelRF );
tg.addChild( wheelLR );
tg.addChild( wheelRR );
tg.addChild( car );
```

Importation de modèles

- Autre exemple : importing3DS_074.mxml

Importation de modèles

- Autre exemple : importing3DS_074.mxml