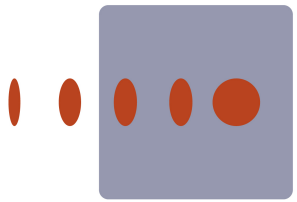
Introduction à VRML

Option I.I.M.

Julien Tierny
julien.tierny@lifl.fr

28 octobre 2007



TELECOMLille1
ECOLE D'INGENIEURS

Menu

- Cours 1 :
  - Généralités ;
  - Le document VRML ;
  - Tracé et positionnement ;
  - Modèle d'éclairement ;
  - Placage de textures ;
- Cours 2 :
  - Ré-utilisation et prototypage ;
  - Interactions ;
  - Animations ;
  - Présentation de X3D.

1. Généralités

VeuReuMeuLeu ?

- VRML : *Virtual Reality Modeling Language* ;
- Langage de description de contenu 3D ;
- Rendu réalisé par un interpréteur VRML (en OpenGL par exemple) ;
- Le *HTML* de la 3D :
  - Intégration de la 3D au Web :
  - Télévente ;
  - Patrimoine historique ;
  - Plate-formes de formation, ...
- Initialement conçu par Silicon Graphics Inc. (bis)
  - Adaptation de l'environnement d'abstraction de scène *Open Inventor* ;
  - *"3D programming for humans"*.

Les atouts de VRML

- Format **ouvert !** (ASCII) ;
- Normalisé par l'ISO :
  - 1994 : Version 1.0 ;
  - 1997 : Version 2.0 ;
- Indépendance de la plate-forme ;
- Nombreux greffons disponibles ;
- Documentation exhaustive ;
- Philosophie : description **hiérarchique** des scènes.

2. Structure d'un document VRML

Philosophie

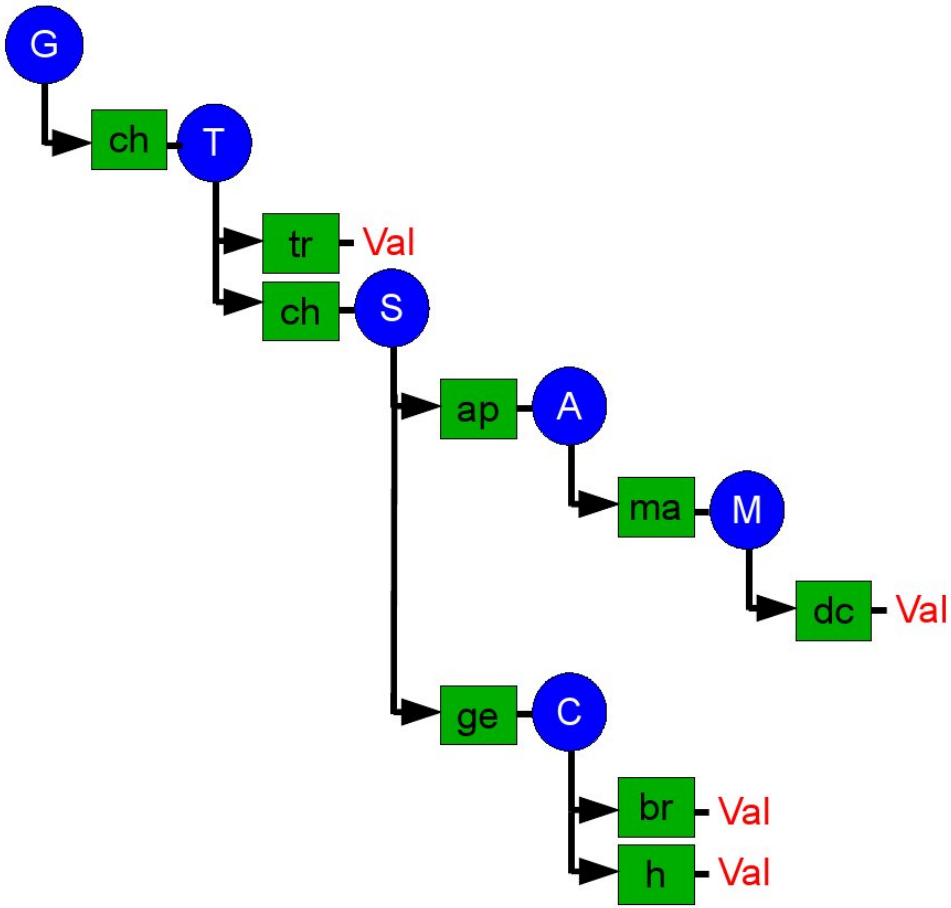
- Focalisation sur le contenu 3D ;
- Modélisation **hiérarchique** de la scène ;
- Représentation par arbre, composé de noeuds ;
- Éléments constitutifs du document :
  - Entête : `#VRML V2.0 utf8` ;
  - Commentaires : caractère joker "#" ;
  - Une hiérarchie de noeuds (en bleu) ;
  - Chaque noeud a des attributs (en vert, typage fort) :
    - Valeurs (en rouge) ;
    - D'autres noeuds.
    - Les attributs ont des valeurs par défaut ;

Exemple de hiérarchie

```
#VRML V2.0 utf8
# Exemple "simple"

Group{
 children[ Transform{
  translation 0 0 -10
  children[ Shape {
   appearance Appearance {
    material Material {
     diffuseColor 0 0 1}}
   geometry Cone{
    bottomRadius 1
    height 2}
  }]
 }]
}
```

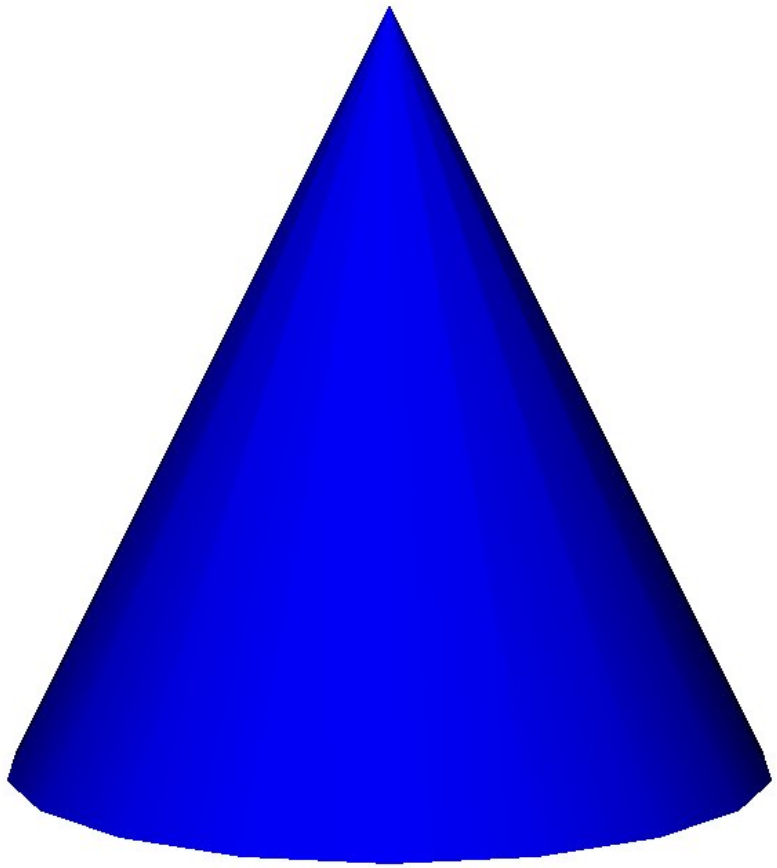
Exemple de hiérarchie

```
#VRML V2.0 utf8
# Exemple "simple"

Group{
 children[ Transform{
  translation 0 0 -10
  children[ Shape {
   appearance Appearance {
    material Material {
     diffuseColor 0 0 1}}
   geometry Cone{
    bottomRadius 1
    height 2}
  }]
 }]
}
```



Exemple de hiérarchie

```
#VRML V2.0 utf8
# Exemple "simple"

Group{
 children[ Transform{
  translation 0 0 -10
  children[ Shape {
   appearance Appearance {
    material Material {
     diffuseColor 0 0 1}}
   geometry Cone{
    bottomRadius 1
    height 2}
  }]
 }]
}
```



Typage

- Quelques types de base :
  - Booléens : `SFBool` ;
  - Vecteurs 2D : `SFVec2f` ;
  - Vecteur 3D : `SFVec3f` ;
- Transformations géométriques :
  - `SFVec3f` : 3 réels (translation et homothétie) ;
  - `SFRotation` : 4 réels, vecteur + angle (en radians !) ;
- Autres :
  - `SFColor` : 3 réels ;
  - `SFString` : une chaine de caractères ;
- Tableaux : :%s/SF/MF/g (crochets, séparés par des virgules) ;
- Mode d'accès : `eventIn` (écriture), `eventOut` (lecture), `exposeField` (lecture/écriture).

3. Tracé et positionnement

Tracé de primitives

- Comme avec GLUT, quelques formes simples sont pré-définies ;
- Attribut `geometry` d'un noeud `Shape` ;
- Noeuds de type `SFGeometry` :
  - Cube : `geometry Box{ size 1 1 1 };`
  - Cone : `geometry Cone{ bottomRadius R, height H, side TRUE/FALSE, bottom TRUE/FALSE };`
  - Cylindre : `geometry Cylinder{ radius R, height H, side TRUE/FALSE, top TRUE/FALSE, bottom TRUE/FALSE };`
  - Sphère : `geometry Sphere{ radius R };`

Ensemble de points

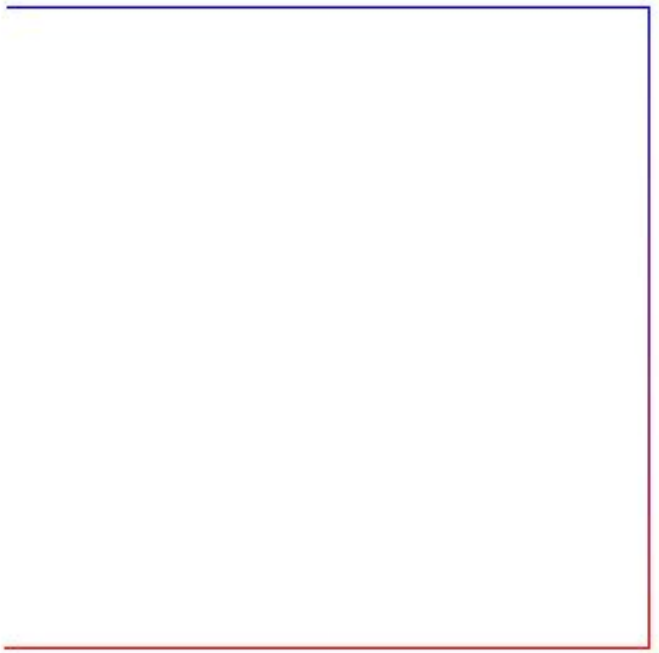
- Attribut `geometry` d'un noeud `Shape` :

```
Shape{
 geometry PointSet {
  coord Coordinate {
   point [
    0 0 0, 1 0 0, 1 1 0, 0 1 0
   ]
  }
 }
}
```

Courbes

- Attribut geometry d'un noeud Shape :

```
Shape{
 geometry IndexedLineSet {
  coord Coordinate {
   point [
    0 0 0, 1 0 0, 1 1 0, 0 1 0
   ]
  }
  color Color{
   color [ 1 0 0, 0 0 1 ]
  }
  coordIndex [ 0,1,2,3 ]
  colorIndex [ 0,0,1,1 ]
 }
}
```



Surfaces

- Représentation par maillages de polygones (cf. cours OpenGL) ;

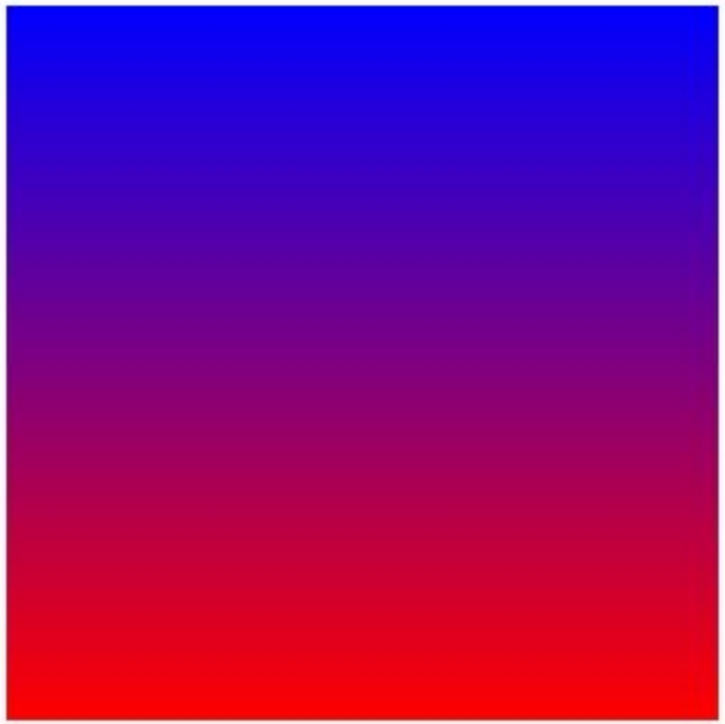
```
Shape{
 geometry IndexedFaceSet {
  coord Coordinate {
   point [
    0 0 0, 1 0 0, 1 1 0, 0 1 0
   ]
  }
  coordIndex [ 0, 1, 2, 3, -1 ]
  color Color {
   color [ 1 0 0, 0 0 1 ]}
  colorIndex [ 0,0,1,1 ]
 }
}
```

- **Exercice :** tetraèdre ;

Surfaces

- Représentation par maillages de polygones (cf. cours OpenGL) ;

```
Shape{
 geometry IndexedFaceSet {
  coord Coordinate {
   point [
    0 0 0, 1 0 0, 1 1 0, 0 1 0
   ]
  }
  coordIndex [ 0, 1, 2, 3, -1 ]
  color Color {
   color [ 1 0 0, 0 0 1 ]}
  colorIndex [ 0,0,1,1 ]
 }
}
```



- **Exercice :** tetraèdre ;

Couleurs indexées

- Permet de définir finement la couleur d'un objet (point par point) ;
- Définition d'une table de couleur :

  color Color { color [1 0 0, 0 0 1]}
- Définition d'un index de sommet (obligatoire si maillage) :

  coordIndex [0, 1, 2, 3, -1]
- Attribution des couleurs sommet par sommet :

  colorIndex [0,0,1,1]

Couleurs indexées

- Permet de définir finement la couleur d'un objet (point par point) ;
- Définition d'une table de couleur :

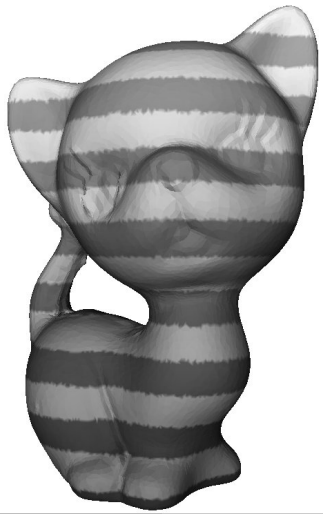
  color Color { color [1 0 0, 0 0 1]}
- Définition d'un index de sommet (obligatoire si maillage) :

  coordIndex [0, 1, 2, 3, -1]
- Attribution des couleurs sommet par sommet :

  colorIndex [0,0,1,1]

Couleurs indexées

- Permet de définir finement la couleur d'un objet (point par point) ;
- Définition d'une table de couleur :
  color Color { color [1 0 0, 0 0 1]}
- Définition d'un index de sommet (obligatoire si maillage) :
  coordIndex [0, 1, 2, 3, -1]
- Attribution des couleurs sommet par sommet :
  colorIndex [0,0,1,1]



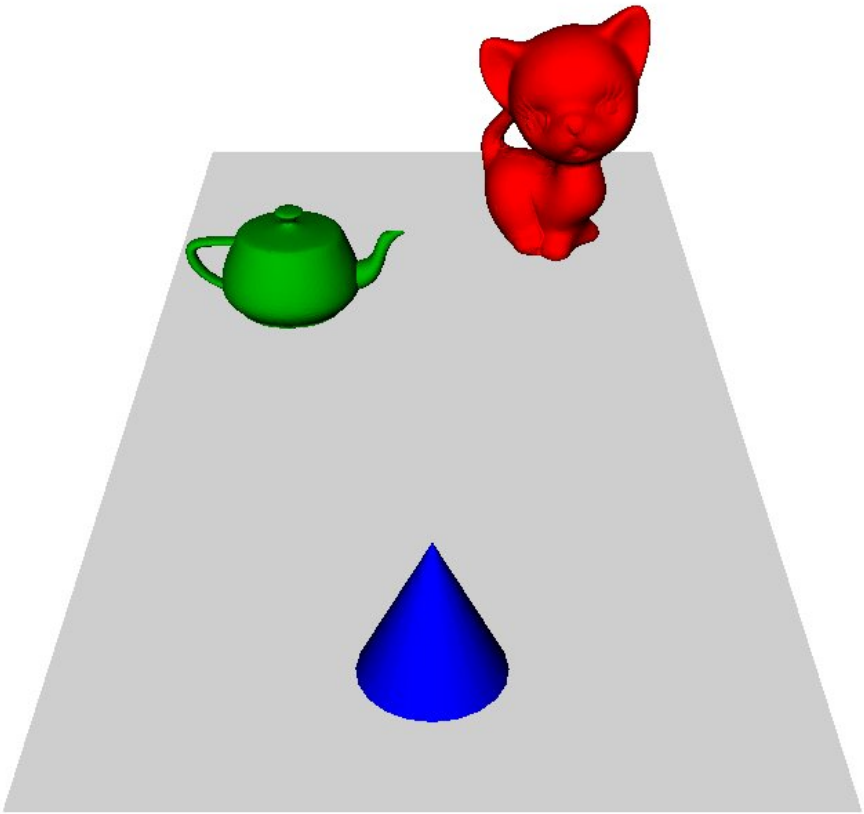
Transformations géométriques

- Positionnement par changements de repère ! (cf. OpenGL)
- Translations, rotations, homothéties.
- Noeud `Transform` ;
- Attributs possibles :
  - `translation` : 3 réels (`SFVec3f`) ;
  - `rotation` : 4 réels (`SFRotation` : vecteur + angle) ;
  - `scale` : 3 réels (`SFVec3f`) ;
  - `children [...]` : un (ou plusieurs) noeud à évaluer après le changement de repère.

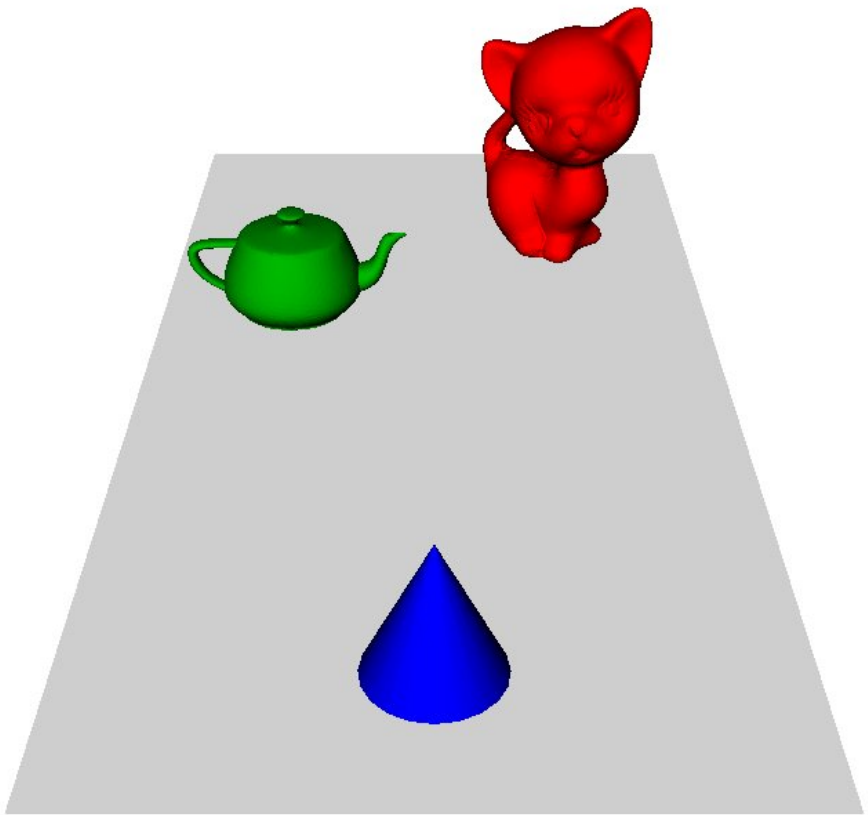
Positionnement hiérarchique

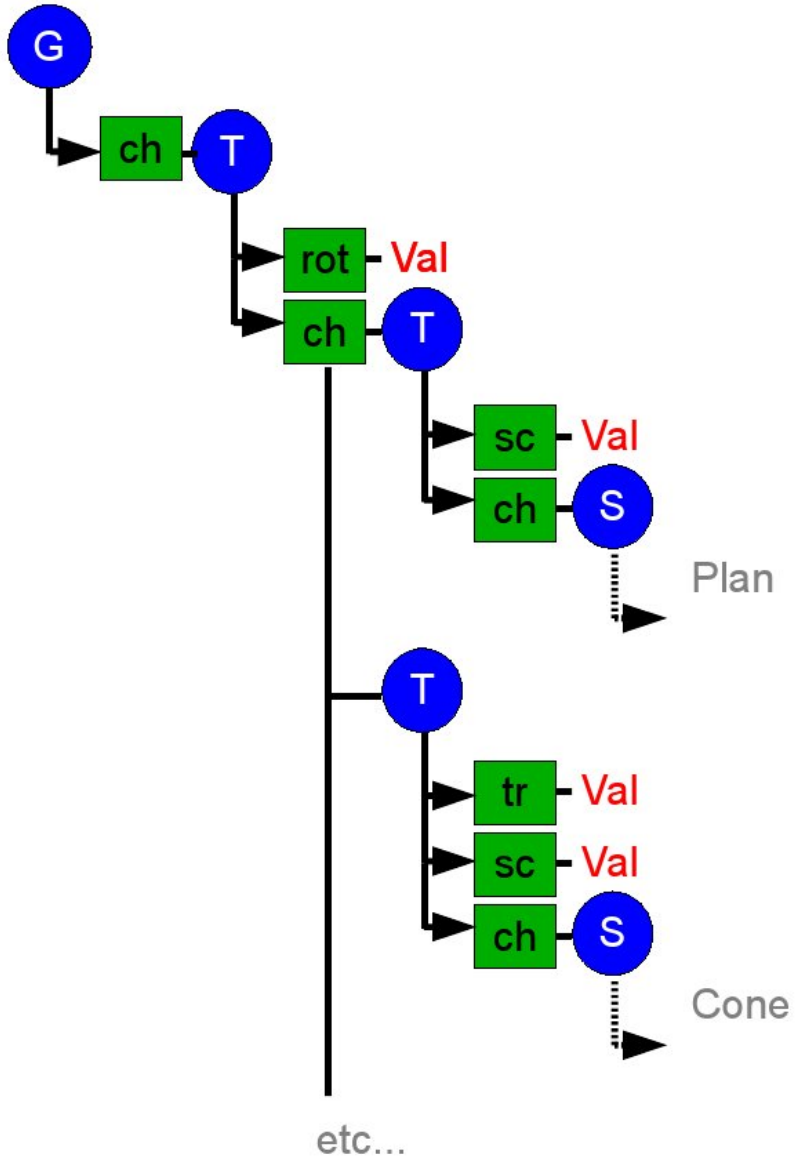
- Disposition **hiérarchique** de la scène !
- Utilisation du noeud `Group` ;
  - Attribut `children [...]` ;
  - Tableau de noeuds (**séparés par des virgules**) ;
  - Permet de grouper les éléments de la scène ;
- Utilisation de l'attribut `children` des transformations également ;
- Mutualisation des transformations ;
- Propagation des transformations dans la hiérarchie ;

Exemple de positionnement



Exemple de positionnement





Dans le fichier VRML

```
#VRML V2.0 utf8
Group {
  children [
    Transform {
      rotation 1 0 0 0.35
      children [
        Transform {
          scale 10 10 20
          children [ Shape { (...Plan...) } ]
        } ,
        Transform {
          translation 0 2 15
          scale 2 2 2
          children [ Shape { (...Cone...) } ]
        } , (...noeuds Transform des autres objets...)
```

3. Modèle d'éclairement

Modèle d'éclairement local

- Ajouter du réalisme en modélisant (localement) les interactions lumineuses ;
- Formulation locale empirique de Phong :
  - Contributions de plusieurs composantes lumineuses.
- Modèle très similaire à celui utilisé par OpenGL ;
- Spécification des caractéristiques lumineuses des matériaux ;
- Spécification des sources lumineuses simplifiées ;

Caractéristiques des matériaux

- Modification de l'apparence d'une forme ;
- Attribut `appearance` d'un noeud `Shape` ;
- Utilisation d'un noeud `Appearance` :
  - Modification du matériau de l'objet ;
  - Attribut `material` d'un noeud `Appearance` ;
  - Utilisation d'un noeud `Material`.
- Attributs d'un noeud type `Material` :
  - `emissiveColor` : contribution ambiante (cf. OpenGL) ;
  - `diffuseColor` : contribution diffuse (cf. OpenGL) ;
  - `specularColor` : contribution spéculaire (cf. OpenGL) ;
  - `ambientIntensity, shininess, transparency` : réflexion, brillance, transparence (1 réel).

Exemple de spécification de caractéristiques lumineuses

```
#VRML V2.0 utf8
Shape{
  appearance Appearance {
    material Material{
      emissiveColor 0 0 0.2
      diffuseColor 0 0 0.6
      specularColor 0 0 0.7
      shininess 0.8
      transparency 0.5
    }
  }
  geometry Box { }
}
```

Sources lumineuses (1/2)

- 3 types de sources (cf. OpenGL) ;
- Attributs communs :
  - `on` : activé ou non ;
  - `color` : couleur émise ;
  - `intensity` : brillance de l'émission ;
  - `ambientIntensity` : intensité de l'émission ambiante.
- Sources directionnelles :
  - Sources à l'infini ;
  - `DirectionalLight { direction 1 0 0 }`

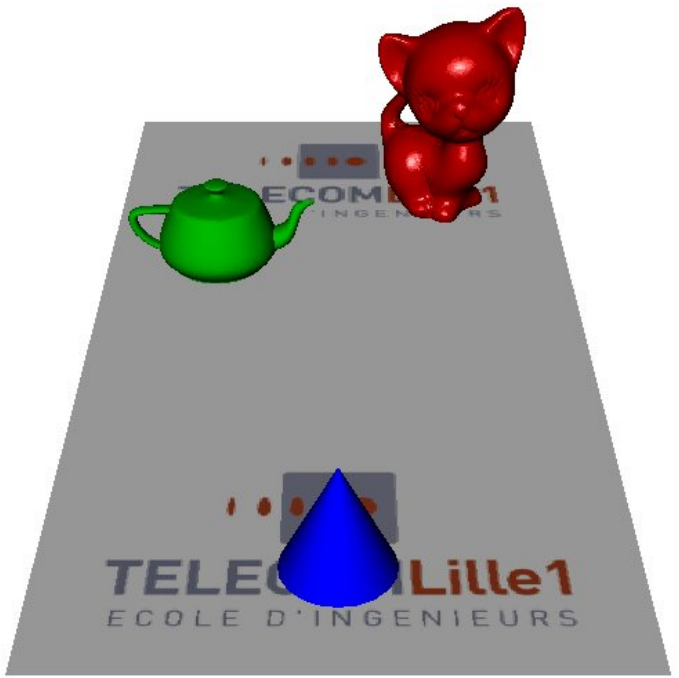
Sources lumineuses (2/2)

- Sources ponctuelles :
  - Émission dans toutes les directions ;
  - ```
    PointLight {
      location 0 0 0
      radius 10
      attenuation 1 0 0
    }
    ```
- Sources de type spot :
  - Source ponctuelle avec émission dans un angle solide donné ;
  - ```
    SpotLight {
      location 0 0 0
      direction 1 0 0
      beamWidth 1.57
      cutOffAngle 0.785
      radius 10.0
      attenuation 1 0 0
    }
    ```

4. Placage de texture

Textures en VRML

- Ajouter du réalisme en "habillant" les géométries ;
- Même démarche qu'en OpenGL ;
- Quelques paramétrisations projectives fournies de base ;
- Correspondance *vertex-texel* ;
- Possibilité d'associer une vidéo à l'espace des textures.



Spécification des textures

- Modification de l'apparence d'une forme :
  - Modification de l'attribut `appearance` du noeud `Shape` ;
- Attribution de la texture au noeud à l'apparence :
  - Modification du champ `texture` du noeud `Appearance` ;
  - Utilisation d'un noeud `ImageTexture` :

```
#VRML V2.0 utf8
Shape {
  appearance Appearance {
      texture ImageTexture {
        url "tl1.jpg"
        repeatS TRUE
        repeatT TRUE
      }
      (...Reste du noeud apparence et shape...)
```

- **Attention :** l'origine de l'espace des textures est en bas à gauche.

Paramétrisation sommet par sommet

- Possible uniquement avec les noeuds `IndexedFaceSet` (attribut `geometry` d'un noeud `Shape`) ;
- Utilisation de la table indexée de sommets ;
- Spécification des coordonnées `s`, `t` dans l'espace des textures pour chaque sommet indexé ;
- Utilisation du noeud `TextureCoordinate` :

```
#VRML V2.0 utf8
Shape {
  appearance Appearance { (Spécification des caractéristiques d'éclairement
et de la texture) }
  geometry IndexedFaceSet {
    coord Coordinate {
      point [ -1 0 1, 1 0 1, 1 0 -1, -1 0 -1 ] }
    coordIndex [ 0,1,2,3,-1 ]
    texCoord TextureCoordinate {
      point [ 0 0, 1 0, 1 1, 0 1 ] }
  }
}
```

5. Ré-utilisation et prototypage

Ré-utilisation de descriptions

- Possibilité de dupliquer des ensembles de géométries ;
- Mots-clefs `DEF` et `USE` ;
- Exemple :

```
#VRML V2.0 utf8
DEF boiboite Box { }
Transform {
  translation 5 0 0
  children [ USE boiboite ]
}
```

- Attention : cet exemple affichera deux cubes !
- Pour ré-utiliser entièrement un autre document :

```
Inline { url "kitten.wrl" }
```

Prototypage

- Possibilité de créer ses propres types de noeuds ;
- Mots-clefs `PROTO` et `EXTERNPROTO` (si ré-utilisation depuis un autre document) ;
- Exemple de définition de noeud (couleur par défaut rouge) :

```
PROTO maboiboite [
  field SFColor couleur 1 0 0
]{
  Shape {
    appearance Apperance {
      material Material {
        diffuseColor IS couleur }}
    geometry Box { }
  }
}
```

- Exemple d'utilisation (avec une couleur bleue) :

```
maboiboite { couleur 0 0 1 }
```

6. Interactions

Gestion des évènements d'interaction par capteurs

- Notion d'évènement d'interacion (producteur/consommateur) :
  - Intervention de l'utilisateur ;
  - Modification de la scène ;
- Détection des évènements d'interaction par *capteurs* (producteurs) :
  - `TouchSensor` : interaction de la souris sur un objet ;
  - `ProximitySensor` : lorsqu'un objet arrive à une certaine proximité de l'observateur ;
  - `VisibilitySensor` : lorsqu'un objet devient visible ;
  - `CylinderSensor` : définition avec la souris de rotations suivant un cylindre ;
  - `PlaneSensor` : gestion des déplacements de la souris dans le plan écran ;
  - `SphereSensor` : définition avec la souris de rotations autour d'une sphère.
- Détection des évènements sur toute la hiérarchie sous-jacente !

Propagation des évènements

- Connexion entre les évènements émis par les capteurs et les noeuds concernés par l'interaction ;
- Identification des producteurs/consommateurs d'évènement par définition (`DEF`) ;
- Utilisation du mot-clef `ROUTE` (`TO`) :
  - Définit la propagation d'un producteur (mode `eventOut`) vers un consommateur (mode `eventIn`) ;
- **Attention** : les champs de `ROUTE` doivent avoir le même type !
- Les connexions `ROUTE` sont placées en fin de fichier.

Exemple d'interaction

```
#VRML V2.0 utf8
Group {
  children [
    DEF Touche TouchSensor { }
    DEF Position Transform {
      translation 0 0 0
      children [ Inline { url "kitten.wrl" } ]
    }
  ]
}
ROUTE Touche.hitPoint_changed TO Position.translation
```

7. Animations

Concept d'animation en VRML

- Animation : modification de la scène ;
- 2 cas de figures :
  - Modifications provoquées par un capteur ;
  - Modifications automatiques à pas de temps réguliers :
    - Utilisation d'un *capteur de temps* : `TimeSensor` ;
- Pour générer les états intermédiaires, utilisation d'un *interpolateur*.

Notion de capteur de temps

- Capteur produisant des évènements en fonction du temps ;
- Déclaration d'un capteur :
  - `DEF MonTimer TimeSensor { }`
- Attribut (à définir, mode `eventIn`) du `TimeSensor` :
  - `cycleInterval` : durée d'un cycle ;
  - `enabled` : activé/désactivé ;
  - `loop` : boucle ou non ;
  - `startTime` : heure de départ du *timer* ;
  - `stopTime` : heure d'arrêt du *timer* ;
- Attributs (sortants, mode `eventOut`) du `TimeSensor` :
  - `isActive` : indique si le chronomètre est lancé ;
  - `cycleTime` : indique l'heure à chaque début de cycle ;
  - `fraction_changed` : fraction du cycle (compris entre 0 et 1) ;
  - `time` : renvoie le temps courant.

Notion d'interpolateur

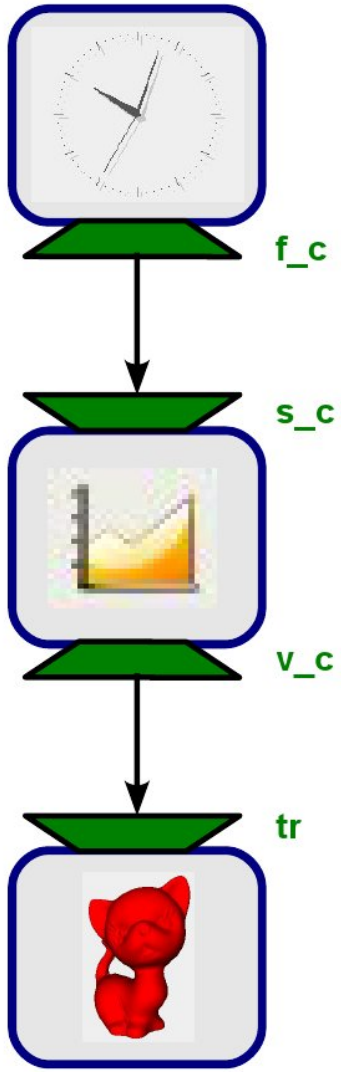
- Permet de modéliser la variation progressive d'état ;
- Plusieurs types d'interpolateurs :
  - `ColorInterpolator` : Modification de couleur ;
  - `PositionInterpolator` : Modification de position ;
  - `NormalInterpolator` : Modification des normales ;
  - `OrientationInterpolator` : Modification de l'orientation ;
  - `ScalarInterpolator` : Modification d'une valeur.
- Système de clés-valeurs :
  - `key` : pas d'interpolation (de 0 à 1) ;
  - `keyValue` : valeurs retournées par l'interpolateur pour chacun des pas d'interpolation ;
- Évènements associés :
  - `set_fraction` (`eventIn`) : positionner l'interpolateur entre 0 et 1 ;
  - `value_changed` (`eventOut`) : valeur interpolée correspondante.

Exemple d'animation avec interpolateur

```
#VRML V2.0 utf8
 DEF MonTimer TimeSensor{
   cycleInterval 5
   loop TRUE
 }
 DEF MonInterpolateur PositionInterpolator{
   key [ 0, 1 ]
   keyValue [ -0.5 0 0, 0.5 0 0 ]
 }
 DEF MonChat Transform{
   translation -0.5 0 0
   children [ Inline { url "kitten.wrl" } ]
 }
 ROUTE MonTimer.fraction_changed
     TO MonInterpolateur.set_fraction
 ROUTE MonInterpolateur.value_changed
     TO MonChat.translation
```

Exemple d'animation avec interpolateur

```
#VRML V2.0 utf8
 DEF MonTimer TimeSensor{
   cycleInterval 5
   loop TRUE
 }
 DEF MonInterpolateur PositionInterpolator{
   key [ 0, 1 ]
   keyValue [ -0.5 0 0, 0.5 0 0 ]
 }
 DEF MonChat Transform{
   translation -0.5 0 0
   children [ Inline { url "kitten.wrl" } ]
 }
 ROUTE MonTimer.fraction_changed
     TO MonInterpolateur.set_fraction
 ROUTE MonInterpolateur.value_changed
     TO MonChat.translation
```



Intégration de langage de scripts

- Utilisation de langages de scripts supportés par les navigateurs Web ;
- Offre davantage de flexibilité au concepteur pour les animations ;
- Création d'un noeud `Script` :

```
DEF MonScript Script{
  eventIn SFTime touchTime
  field SFBool Active TRUE
  eventOut SFBool enabled
  url [
    "javascript :
    function touchTime(eventValue) {
      if( !Active){
        Active=true ;
        enabled=true ;
      } else {
        enabled=false ;
        Active=false ;
      }
    }"
  ]
}
```

Spécification du script

- Utilisation d'évènements pour modéliser les entrées/sorties ;
- Prototypage :
  - `eventIn` : entrée (une fonction par entrée) ;
  - `eventOut` : valeur retournée par le script ;
  - `field` : variable locale ;
- Utilisation du script :
  - Connexion entre producteurs et consommateurs d'évènements par des `ROUTE`.

Mais aussi... rendu sonore

- Pour mettre de la musique :)
- Permet de modéliser (au niveau spatial) le son émis par des éléments d'une scène ;
- Exemple :

```
Sound {
  source DEF MonSon AudioClip {
    url "son.wav"
  }
  direction 1 0 0
  location 0 0 0
  minBack 0
  maxBack 5
  minFront 0
  maxFront 10
}
```

- Exemple d'utilisation :

```
ROUTE MaTouche.touchTime TO MonSon.startTime
```

8. Présentation de X3D

Le futur de VRML

- VRML est le HTML de la 3D ;
- X3D (*eXtensible 3D*) est le XML de la 3D ;
- Normalisé par l'ISO ;
- Évolution XML du VRML (+ nouvelles fonctionnalités : *H-Anim*, multi-utilisateurs, ...) ;
- Meilleure intégration avec les technologies Web récentes ;
- Alternatives :
  - Collada (Khronos Group) ;
  - U3D ;
  - 3DXML.

Exemple de scène au format X3D

```
<X3D>
  <Scene>
    <Shape>
      <Sphere radius="2.2"/>
      <Appearance>
          <Material ambientIntensity="0.4" diffuseColor="0.3
0.3 1.0"/>
      </Appearance>
    </Shape>
  </Scene>
</X3D>
```

Ré-utilisation en X3D

```
<Transform translation='2.0 0.0 0.0'>
  <Shape DEF='Joe'>
    <Sphere radius='0.2'/>
  </Shape>
</Transform>
<Transform translation='-2.0 0.0 0.0'>
  <Shape USE='Joe'/>
</Transform>
```

Prototypage et X3D

```
<ProtoDeclare>
  <ProtoInterface>
    <field name='legColor' type='SFColor' value='.8 .4 .7'
      accessType='initializeOnly'/>
    <field name='topColor' type='SFColor' value='.6 .6 .1'
      accessType='initializeOnly'/>
  </ProtoInterface>
  <ProtoBody>
  ...
  </ProtoBody>
</ProtoDeclare>
<ProtoInstance name='TwoColorTable'>
  <fieldValue name='legColor' value='1 0 0'/>
  <fieldValue name='topColor' value='0 1 0'/>
</ProtoInstance>
```

Animation en X3D

```
<X3D version='3.0'>
  <Scene>
    <Transform DEF='XForm'>
      <Shape>
        <Box/>
        <Appearance>
          <Material diffuseColor='1.0 0.0 0.0'/>
        </Appearance>
      </Shape>
      <TouchSensor DEF='Clicker'/>
      <TimeSensor DEF='TimeSource' cycleInterval='2.0'/>
      <OrientationInterpolator DEF='Animation'
        keyValue='0.0 1.0 0.0 0.0, 0.0 1.0 0.0 2.1,
        0.0 1.0 0.0 4.2, 0.0 1.0 0.0 0.0'
         key='0.0 0.33 0.66 1.0'/>
    </Transform>
    <ROUTE fromNode='Clicker' fromField='touchTime' toNode='TimeSource'
      toField='startTime'/>
    <ROUTE fromNode='TimeSource' fromField='fraction_changed' toNode='Animation'
      toField='set_fraction'/>
    <ROUTE fromNode='Animation' fromField='value_changed' toNode='XForm'
      toField='rotation'/>
  </Scene>
</X3D>
```

Conclusion

- Entités de haute abstraction pour la modélisation d'interactions et d'animations ;
- Focalisation sur la description de contenu ;
- Création de contenu relativement aisée ;
- Bonne intégration avec les technologies Web ;
- ... mais format verbeux ;
- Décollage imminent...