MASTER INFORMATIQUE-STL 1

AGM- COURS-6

Animation élémentaire avec VRML

- Animation par routage
  - capteurs
  - interpolateurs
  - événements
- Animation par script
  - modification d'élément feuille
  - ajout d'éléments à un noeud interne Transform
- Prototype + script
  - pour calculer des valeurs / créer des noeuds
  - pour animer certaines instances

ANIMATION PAR ROUTAGE

Animation : idée liée au temps

Changer à l'*instant* t de *valeur* v typée dans le nœud N, attribut A

- instant t, 2 choix
  - o provenant d'une action utilisateur (TouchSensor, PlaneSensor)..
  - calculé à partir de la durée écoulée à partir d'un temps initial (TimeSensor + horloge système)
- valeur v typée dans attribut A du nœud N, 2 choix
  - o valeur d'un attribut A'd'un nœud N'
  - o valeur du type souhaitée calculée par interpolation linéaire en fonction de la durée écoulée etc… (XtypeXInterpolator)

Graphe acyclique G

- Chaque transformation souhaitée est notée par un arc reliant N'.A 'à N.A, capable de transmettre la donnée de N à N'
- L'ensemble des arcs forme un graphe acyclique G

Cascade d'événements et Parcours de G : un *événement initial* (clic souris, alarme sytème) sur un nœud N(voir plus loin) à l'instant ***t*** déclenche des transformations décrites dans G à partir de N; le changement de certaines données engendre un *événement* et le parcours d'autres arcs. Soit un *parcours* (fini) en cascade d'événement marqués par ***t***.

Événements

Événement : donnée – datée - typée

- ✫initial (provenant d'un capteur (*sensor*))
- ✫transmis (et traité) en cascade au long d'un graphe acyclique, à partir d'un événement initial :
- ✫pour chaque arc, l'événement est
  - émis par un nœud, valeur d'un attribut (initial, provenant de l'extérieur, ou calculé)
  - reçu par un nœud, la valeur redéfinit un attribut

Traitement :

- ✫transformation d'un attribut du noeud recevant la valeur, et d'autres par calcul
- ✫changement (éventuel) dans l'affichage tenant compte du changement (de translation, de rotation, de couleur, de coordonnées)..
- ✫et/ou émission d'un autre événement (transmission d'une nouvelle valeur) vers un autre nœud du graphe, *s'il existe un arc sortant* utilisant le nom d'un attribut (champ) modifié par le traitement.

GRAPHE DE TRANSMISSION DES ÉVÉNEMENTS: **ROUTE**

Chaque arc est décrit textuellement :

Instruction ROUTE, construit un arc entre (champs de) deux nœuds.

Autant d'instructions que d'arcs dans le graphe

Deux écritures équivalentes :

```
ROUTE NOMNOEUD1.nomchamp1 TO
                NOMNOEUD2.nomchamp2
ROUTE NOMNOEUD1.nomchamp1_changed TO
                NOMNOEUD2.set_nomchamp2
```

La seconde est plus explicite et indispensable pour la traduction en X3D : l'arc créé est parcouru dès que la valeur est changée; la valeur source transportée redéfinit le champ but.

Rappel de la classification des champs de noeuds

- **`field` : non modifiable (par routage) après création**
- **`eventIn`**
- **`eventOut`**

`eventIn` et `eventOut` : pas de valeur par défaut

- **`exposedField` : valeur par défaut + eventIn +eventOut**

Un 'field' est modifiable par programme javascript (ou java)

Un champ *source* est `eventOut` ou `Exposed field`
Un champ *but* est `eventIn` ou `exposed field`

Exemple de graphe:

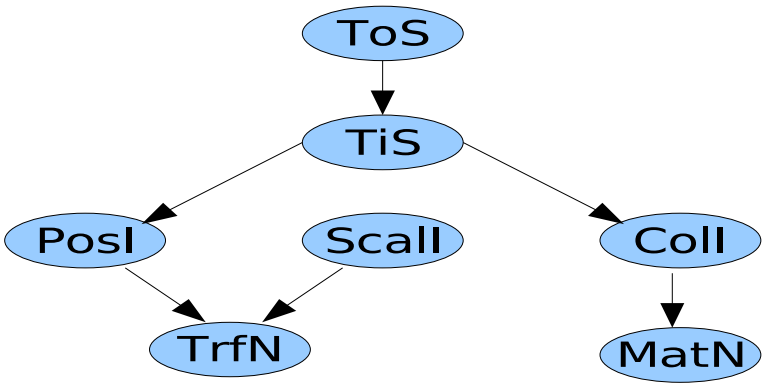
Les nœuds sont obligatoirement nommés (par DEF)

```
ROUTE ToS.touchTime TO TiS.startTime
ROUTE TiS.fraction_changed TO ColI.set_fraction
ROUTE TiS.fraction_changed TO PosI.set_fraction
ROUTE TiS.fraction_changed TO ScalI.set_fraction
ROUTE PosI.value_changed TO TrfN.set_translation
ROUTE ScalI.value_changed TO TrfN.set_scale
ROUTE ColI.value_changed TO MatN.set_diffuseColor
```

Remarque : TrfN reçoit 2 arcs, *pour 2 champs différents*

Les sommets sont de genre :

- ☆ Sensor : peut recevoir des événements de l'environnement (utilisateur, système) ToS,TiS
- ☆ Interpolateur : c'est un convertisseur entre type de valeur (flottant vers une couleur ou une rotation)… PosI, ColI, ScalI
- ☆ Nœud usuel de description de la scène MatN, trfN ==> transformation visible à l'affichage



Interpolateurs –convertisseurs

Ils reçoivent une valeur flottante et la transforment en une valeur dŭ n type structuré

`fraction` : une valeur flottante entre min et max

`key` : un tableau de n (n > 3) flottants (valeurs croissantes entre min et max, souvent entre 0 et 1)

`keyValue` : un tableau de n valeurs du type à obtenir

`value` : la valeur convertie calculée par interpolation à partir de fraction

Les champs les plus utilisés dans l'animation :
`set_fraction` par un arc entrant
`value_changed` calculée par une interpolation sur les arcs sortant.

```
YYYYInterpolator{
eventIn SFFloat set_fraction
exposedField MFFloat key [ ]
exposedField MFXXX keyValue [ ]
eventOut  SFXXX value_changed}
```

Exemple

```
DEF Cli ColorInterpolator {
    key [ 0 0.5 1]
    keyValue [ 1 .5 1, 1 1 .5, 1 .5 1 ]
    }
```

Pour la fraction 0.25 , entre 1 .5 1 (violet clair) et 1 1 .5 (jaune pale) , donne la couleur 1 0.75 0.75 (rose pale).

intervalle par intervalle -> variations de rythme (changement lent /rapide).
Pour une animation cyclique : dernière valeur convertie = première pour une animation en boucle fluide .

Liste des 6 interpolateurs disponibles

value_changed est **SFxxx**, **keyValue** est **MFxxx**

- **SFColor ColorInterpolator #(S/M)FColor**
- **SFVec3f PositionInterpolator # (S/M)FVec3FSFFloat : translation**
- **ScalarInterpolator # (S/M)FFloat**
- **SFRotation OrientationInterpolator # (S/M)FVec3F** : champ rotation

Pour les IndexedFacedSet :

value_changed est aussi MFVec3F,
keyValue est MFVec3F, n_valclés * nb_points

- **MFVec3f CoordinateInterpolator #(S/M)FVec3F**
- **MFVec3f NormalInterpolator # (S/M)FVec3F**

keyValue est MFVec3F, n_valclés * nb_faces

Capteur-Sensor : récepteur-émetteur

- Les *capteurs (Sensors)* peuvent *recevoir* sur les EventIn ou exposed field:
  -- des événements externes (le temps système, un clic utilisateur)
  -- des événements-valeurs provenant d'autres sensors par **ROUTE**
- Ils peuvent *émettre* des événements-valeurs par eventOut ou exposed field vers d'autres capteurs ou vers des interpolateurs

Tous ont les champs :

exposedField enabled (SFBool)
basculer à TRUE pour mettre en fonction (y reste), à FALSE pour désactiver

eventOut isActive (SFBool) est émis TRUE ou FALSE

Contrainte : enabled -> faux, alors isActive -> faux.

LISTE DES SENSORS

Pour évoluer au cours du temps :

- **TimeSensor**

Durée de l'animation :

```
exposedField SFTime cycleInterval
```

La proportion (entre 0 et 1) du cycle écoulé :

```
eventOut fraction_changed
```

Souris

- **TouchSensor** `SFTime TouchTime`, (lors d'un clic)
  `SFBool isOver`, (arrivée pointeur souris)
- `PlaneSensor` `SFVec3f offset, translation_changed`
- `SphereSensor`
- `CylinderSensor`

Changement de point de vue

- `ProximitySensor`
- `VisibilitySensor`
- `CollisionSensor`

Exemple 1 de routage

Mise en valeur d'un élément, deux images en bascule, indépendant du temps, suivant déplacement de la souris par lut ilisateur

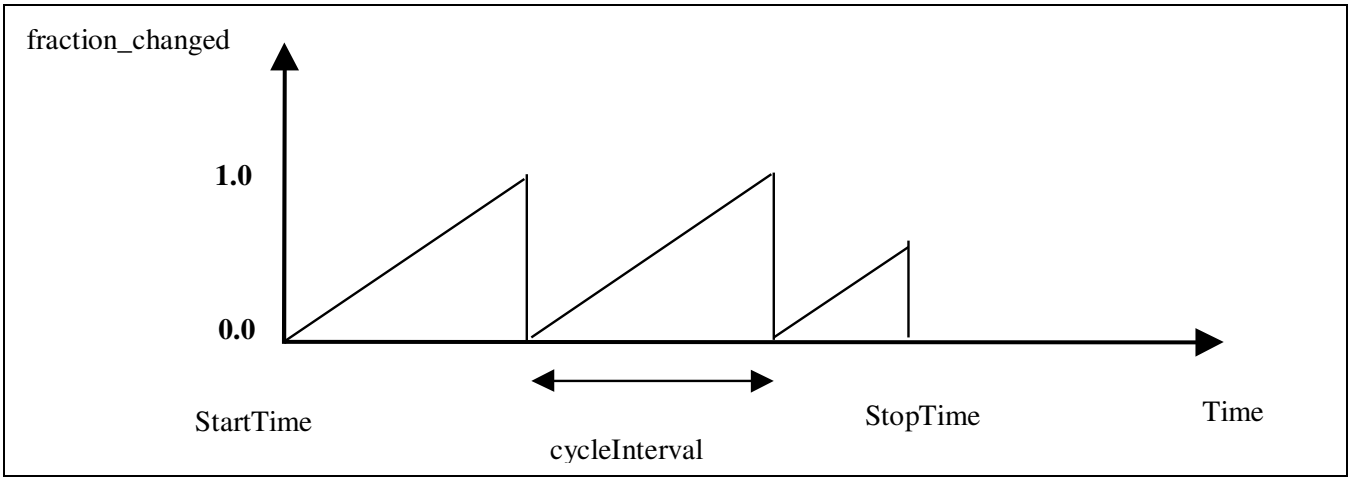
```
Transform {
    children [
#une lampe qui peut sallumer
       DEF PL PointLight {
             on FALSE
             location  3 3 3
             color  0.95 0.95 0.95
          }

       DEF TS TouchSensor {}
          Shape {
                  geometry Box {}
                  appearance Appearance {material
Material { diffuseColor 0.3 0.3 0.3
          }}}
           ]
        }

ROUTE TS.isOver TO PL.on
```

DESCRIPTION TIME SENSOR

C'est le nœud central de toute animation !

```
TimeSensor{
    exposedField SFTime cycleInterval 1
     // en secondes >0
    exposedField SFBool enabled TRUE
    exposedField SFBool loop FALSE
    //refaire en fin de cycle ?
    exposedField SFTime startTime 0
    //défini au démarrage si enabled TRUE,
// par routage depuis TouchSensor ou autre TimeSensor
    exposedField SFTime stopTime 0
    //pas utilisé si < startTime
    eventOut SFTime cycleTime
    //instant de début du cycle en cours
    eventOut SFFloat fraction_changed
    // proportion écoulée du cycle en cours
    eventOut SFBool isActive
    //vrai démarrage -> (fin ou enabled FALSE)
    eventOut SFTime time
    //date de l'instant (du systeme)
}
```

fraction ≅ (time –cycleTime)/cycleInterval



DESCRIPTION DE TOUCHSENSOR,

```
TouchSensor{
    exposedField SFBool enabled TRUE
    eventOut SFVec3f hitNormal_changed
    eventOut SFVec3f hitPoint_changed
    eventOut SFVec3f hitTexCoord_changed
    eventOut SFBool isActive
    eventOut SFBool isOver
    eventOut SFTime touchTime
}
```

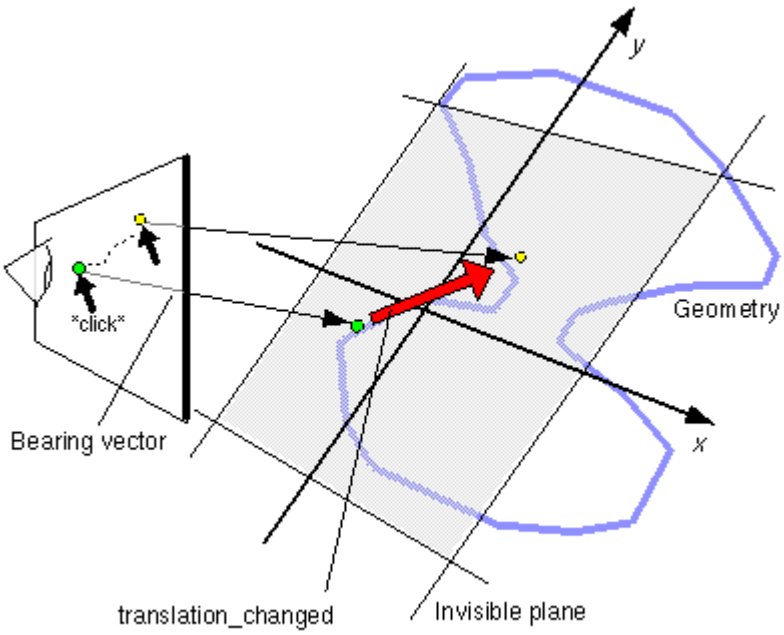
Lorsqu'un TouchSensor est descendant d'un nœud, tous les descendants de ce nœud sont 'sous surveillance' : un curseur spécial apprait dès que la souris est sur l'image d'un descendant et l'événement isOver est émis (si enabled TRUE).

En cas de clic,
`touchTime` permet de démarrer une animation

```
PlaneSensor{
    exposedField SFBool autoOffset     TRUE
    exposedField SFBool enabled        TRUE
    exposedField SFVec2f maxPosition 1 -1
    exposedField SFVec2f minPosition  0 0
    exposedField SFVec3f offset       0 0 0
    eventOut SFBool isActive
    eventOut SFVec3f TrackPoint_changed
    eventOut SFVec3f translation_changed
}
```

`minPosition` et `maxPosition` définissent un rectangle (vide par défaut) dans *le plan Z=0 du repère local* ; où la coordonnée du point courant définit l'événement de sortie `translation_changed` utilisable pour déplacer un nœud de la scène.

Schéma extrait de Carey,Bell : Annotated VRML97 Reference Manual
Décrit les coordonnées délivrées par le PlaneSensor dans un plan z=0 du repère local non parallèle au plan de vue



Exemple 2 de routage

Animation dépendant du temps : correspond au graphe page 4.
Un bloc change de couleur, de taille, de position en 4 secondes, animation démarrée par TouchSensor.
Les noms des sensors sont terminés par S, ceux des interpolateurs par I, ceux des autres noeuds par N.

```
DEF TrfN Transform {
    children [
        DEF ToS TouchSensor {}
        DEF ColI ColorInterpolator {
            key [ 0 0.5 1]
            keyValue [ 1 .5 1, 1 1 .5, 1 .5 1 ]
        }
        DEF PosI PositionInterpolator {
            key [ 0 .25 .5 0.75 1]
            keyValue [ 0 0 0, 1 1 0, -1 -1 1, 1 0 -1, 0 0 0]
        }
        DEF ScalI PositionInterpolator {
            key [ 0 .25 .5 0.75 1]
            keyValue [ 1 1 1, 0.3 0.3 0.3, 0.5 0.5 0.5,
                       0.75 0.75 0.75, 1 1 1]
        }
        Shape {
            geometry Box {}
            appearance Appearance {
                material DEF MatN Material {
                    diffuseColor 0.5 0.5 0.5}}}
        ] # Fin children
}
DEF TiS TimeSensor {
    cycleInterval 4
}

ROUTE ToS.touchTime TO TiS.startTime
ROUTE TiS.fraction_changed TO ColI.set_fraction
ROUTE TiS.fraction_changed TO PosI.set_fraction
ROUTE TiS.fraction_changed TO ScalI.set_fraction
ROUTE PosI.value_changed TO TrfN.set_translation
ROUTE ScalI.value_changed TO TrfN.set_scale
ROUTE ColI.value_changed TO MatN.set_diffuseColor
```

Exemple 3 de routage

Saisie de coordonnées et déplacement d'un objet :

On reste dans le plan Z=0 ; seule la direction x est prise en compte.

```
Viewpoint{ position 0 0 20}
Transform{
    children [
        DEF CUBE Transform{
            children Shape { geometry Box{}}
        }
        DEF XCOORD_S PlaneSensor {
            minPosition -5 0
            maxPosition 5 0
         }
    ]
}
# Routage
ROUTE XCOORD_S.translation_changed TO  CUBE.set_translation
```

L'action commence quand on *clique sur un descendant affichable du père du sensor* (le seul cube ici), jusqu'au relachement. Seules les coordonnées X entre −5 et +5 sont prises en compte, quel que soit Y.

Comment changer une valeur entière ?

Switch : basculer rapidement en changeant juste l'entier.

```
Switch{
    exposedField MFNode   choice       [ ]
    exposedField SFInt32  whichChoice  -1
}
```

Exemple 4 :

on voit uniquement le feu rouge (initial) ou le feu vert (quand ?), jamais les deux :

```
DEF FeuxSWN Switch {
    whichChoice 0
    choice [
        #0 affiche feu rouge
        Transform {
            translation 0 1 0
            rotation  1 0 0 1.57
            children  Shape {
                geometry DEF C Cylinder {radius 0.5}
                appearance Appearance {
                    material Material {
                        diffuseColor 1 0 0 }}}
        #1 affiche feu vert
        Transform {
            translation 0 -1 0
            rotation  1 0 0 1.57
        children Shape {
                geometry USE C
                appearance Appearance {
                material Material {
                        diffuseColor  0 1 0 }}}
          }
        ]
      }
```

- Mais comment changer **whichChoice ?**
  - Pas d'interpolateur à valeur entière, pas de champ SFInt facile à utiliser dans les Sensors.
- Autre outil : les *scripts*.

SCRIPTS : Le nœud Script

Le nœud **`Script`** est un nœud VRML capable

- d'associer un *traitement spécifique à un événement reçu* d'un autre noeud,
- *de construire par calcul une (ou plusieurs) valeur(s)* d'autres types
- pour les *transmettre comme événements* à d'autres nœuds.

Il permet dIn sérer un traitement autre que le traitement standard dans le sommet du graphe de routage, en particulier de faire des calculs.

`Script` *est le seul nœud dont le nombre de champs nommés n'est pas fixé*.

C'est ainsi que sont réalisés les interpolateurs, et les `Box`, `Cones` .q ui construisent des `IndexedFacedSet`.

Le script a *deux parties, une entête VRML, une de code ; il est fait pour intervenir dans le graphe de routage : événements*.

Le script décrit :

- les événements entrants qu'il sait traiter
- les événements sortants
- les `fields` : variables partagées entre plusieurs fonctions
- le champ essentiel `url` de type `MFString` : chaîne du

code incluse dans le nœud ou url nommant un fichier contenant ce code. Décrit les *traitements* des événements entrants et des fonctions auxiliaires.

Les événements sont destinés au routage.
Les `exposedField` sont interdits (en dehors du champ `url`).

L'entête

```
Script {
exposedField MFString url
    field SFBool directOutput FALSE
    field SFBool mustEvaluate FALSE

    # liste de champs (field, eventIn, eventOut
    # ajoutés par le concepteur)
    …
}
```

Les traitements

Les traitements sont des fonctions (EMCAscript ici) portant le nom des champs `eventIn` que le script déclare (ou a par défaut). (EMCAscript = javascript standardisé)

Deux traitements particuliers :

- `initialize()`, lors du chargement
- `shutdown()`, lorsque la page est déchargée ou le nœud du script détruit

Les autres traitements portent le nom des champs `eventIn` que le script déclare : une telle fonction est alors un événement (en réalité la réponse automatique préenregistrée à un événement) accessible (exécutable automatiquement) par routage; elle peut calculer des valeurs, modifier des champs.

Le fonctionnement du routage :

- L'arrivée par routage d'un `eventIn` déclenche le traitement associé.
- Le changement par un traitement d'un `eventOut` déclenche son envoi au routage qui le concerne.

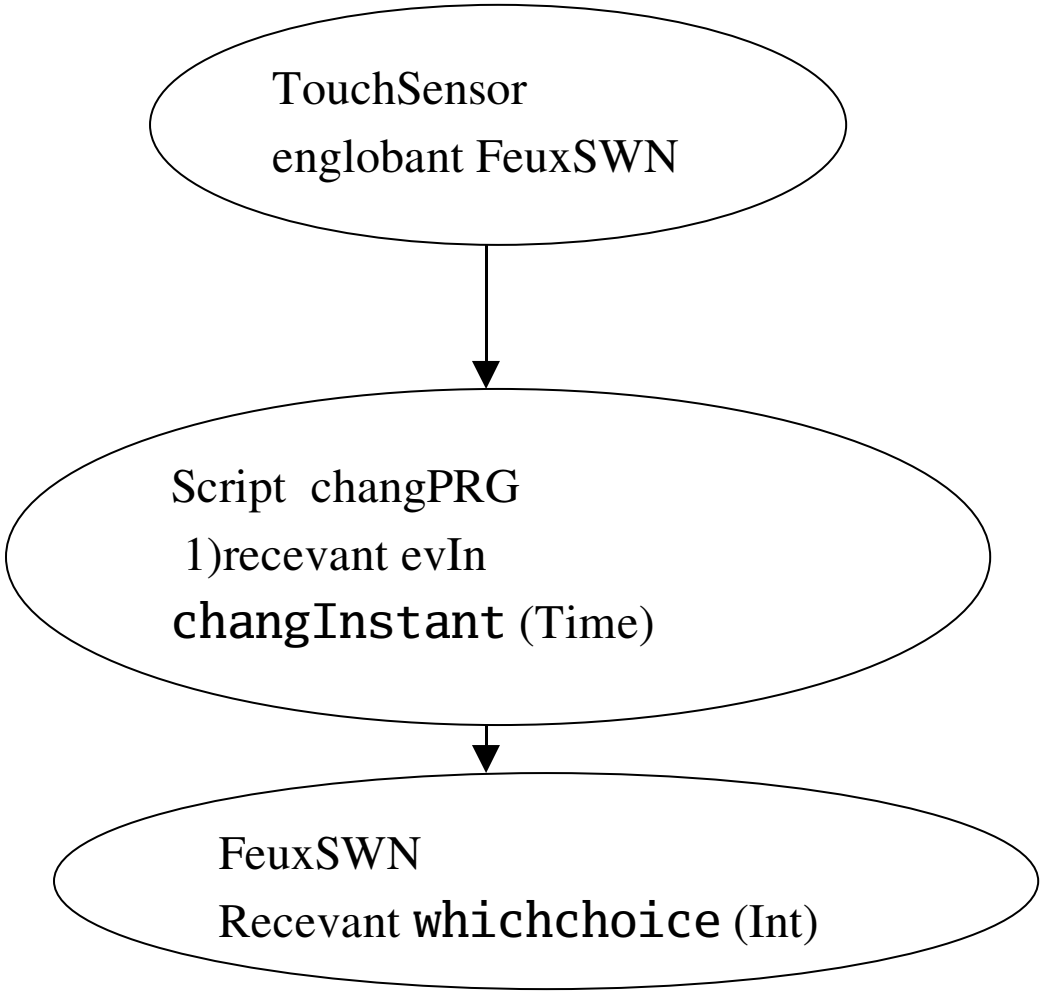
Exemple :

```
Script {
 #partie VRML
 #les événements nécessaires au routage
 eventIn nomEvIn
 eventOut nomEvOut

 #les champs conservant valeurs initiales ou #
 #calculées (entre 2 traitements d'événements
 field nomtype nomchamp valpardefaut

 #les traitements d'événements entrant
 url "javascript:
   //commentaire du code, proto fixe.
   function nomEvIn(value, time){
     …
     eventOut = nouvVal
    …
   }
   //liste d'autres fonctions
   …
 "  # fin dernière fonction et de EMCAscript

 } #fin du nœud script (VRML)
```

Lù rl est soit une chaîne EMCAscript (seul cas envisagé dans ce cours), soit lù rl d'un fichier EMCAscript ou dù ne classe Java contenant les traitements.

Exemple 5 :

Modifier la couleur du feu FeuxSWN (englobé dans un touchSensor) par clic souris.



Deux arcs, d'où deux instructions de routage :

```
ROUTE ToS.touchTime TO ChangP.changeInstant
ROUTE ChangP.choix TO FeuxSW.whichChoice
```

Avec la variante toujours possible:

```
#ROUTE ToS.touchTime_changed TO
          ChangP.set_changeInstant
#ROUTE ChangP.choix_changed TO
          FeuxSW.set_whichChoice
```

Le script **ChangPRG** :

```
 DEF ChangPRG Script {
    #commentaire VRML, nœuds à valeur par défaut
    eventIn SFTime changeInstant
    eventOut SFInt32 choix # lié au switch
    field  SFBool  vert FALSE
    url "javascript:
        //commentaire du code
        function changeInstant(value, time) {
           // permute les valeurs de choix
           vert = !vert;
           if (vert) { choix = 1; }
           else      { choix = 0; }
        }"
}
```

Le résumé du fichier :

feu alterné :

```
Transform {
   children [
        DEF ToS TouchSensor {}
        DEF FeuxSWN Switch {cf exemple4}.
        DEF ChangPRG Script {cf au dessus}.
   ]
}

ROUTE ToS.touchTime TO ChangP.changeInstant
ROUTE ChangP.choix TO FeuxSW.whichChoice
```

EMCAscript (aka Javascript)

Langage simple, non typé, interprété, enrichi de classes et d'objets par le visualisateur.

Les structures de contrôle : celles de C-Java.
Les booléens se notent `true` et `false` (`TRUE` et `FALSE`), not se note ! etc…

Les *variables ne sont pas* nécessairement *typées*, ni les valeurs de retour des fonctions ; il est donc facile d'écrire des choses incohérentes, mais aussi très facile d'écrire quelques lignes.

Les *valeurs sont typées* (entier, flottant, chaine, type-classe nommé structuré).

La *concaténation de chaînes* se note + comme en java,
les nombres sont implicitement convertis en chaîne dans un contexte chaîne.

Les *chaînes de caractères* alternativement délimitées par " " et ' ' , le caractère d'échappement étant \.

```
chaine = 'l\'entier "n" vaut ' + n;
```

Un *sous programme* est noté classiquement :

```
function(< paramètres nommés, sans type>)
{<corps> }
```

Pas de contrôle de la mémoire : instantation avec l'opérateur new, pas de libération de la mémoire (garbage collecting) à la manière de Java.

Les objets structurés

Instantier de nouveaux objets de classe connue :

```
translation= new SFVec3f(2.0,2,2);
//c'est un tableau, index 0 à 2
rouge = new SFColor(1.0, 0.0, 0.0) ;
```

Accéder aux champs des instances de ces classes :

```
translation[2]=3;
```

Les méthodes *fonctionnelles :*

```
trsl2=translation.add(new SFVec3f(2.0,2,2))
```

retourne un nouvel objet de valeur 4,4,4 mais ne change pas `translation`, pas un langage objet classique !

Noms de classe connus `SFColor`, `SFRotation`, …

Les Tableaux

Tableau avec le constructeur `MFXXX` et une suite de `SFXXX` correspondant, séparés par des virgules. Sont de longueur variable : Exemple, un tableau de couleurs

```
tabcouleurs = new MFColor(new SFColor(1,0,0),
new SFColor(1,1,0),new SFColor(0,1,0));

n=tabcouleurs.length //vaut provisoirement 3

tabcouleurs[4] = rouge ;
n=tabcouleurs.length //vaut 5,
//tabcouleurs[3] est null
```

Modification directe de nœuds existants

Le visualisateur reçoit une description textuelle de la scène. Il construit à partir de là une représentation mémoire de l'arbre de la scène, au travers d'OpenGL le plus souvent (ou de Java3D).

A partir de cette représentation, il affiche la scène.

Les interpolateurs (couplés à des capteurs) modifient certains champs de certains noeuds et le visionneur réaffiche la scène à partir de cette représentation mémoire modifiée.

Le but de cette partie est de réaliser ces modifications par des fonctions de script écrites par le concepteur de la page pour ne pas être limité aux effets prédéfinis par routage.

Le DOM (document objet model) de VRML a le rôle des interfaces java (mais il n'est pas lié à un langage spécifique) : il définit les types, champs, fonctions disponibles pour manipuler les objets de la représentation mémoire. C'est une représentation abstraite arborescente, indépendante de celle qui existe dans la réalité : le visualisateur doit fournir une réalisation concrète de cette interface en fonction de ses choix.

Le DOM de HTML permet de manipuler de manière similaire un document afficher dans une page pour changer son aspect après l'affichage initial, en dehors des vidéos ou GIF animés.

Dans ce cadre, un nœud est une *référence d'objet* et un nom défini par DEF donne un accès à une référence sur un noeud de l'arbre (sans être obligé de parcourir à partir de la racine pour atteindre le 3ème fils du 2eme fils de la racine pour un cas simple...).

Attention, pour que les changements se reflètent immédiatement dans l'affichage, le champ `directOutput` du script doit être `TRUE;` le visualisateur réaffiche alors la scène.

Pour modifier un nœud, il faut accéder à un noeud nommé dans l'entête, se déplacer dans ses descendants et/ou au champs par javascript.

- Déclarer dans un script des paramètres (type SFNode) prenant comme valeur un noeud nommé par DEF créé auparavant dans le fichier VRML, grâce au mot clé *USE*:

```
Script{
    field      SFNode    nom_scr USE nomN
```

- `nomN` d'un noeud, défini par DEF dans la partie VRML
- `nom_scr` variable connue dans la partie javascript
- mis en correspondance au *chargement* du script : ce n'est pas un paramètre valeur comme une couleur, mais une *référence constante* sur un objet (dont on peut modifier les champs).
- syntaxe x3d:

```
<field name="TransformPourPion_scr"
 type="SFNode" accessType="initializeOnly">
      <Transform USE="TransformPourPion"/>
 </field>
```

- Accès au 2ème fils (si nom_scr est de type Transform):

```
fils2 = nom_scr.chidren[1]
```

- Modifier un champ d'une référence `nom_scr` connue dans le script

```
nom_scr.nomchamp = val
ou fils2.translation.x = 10
```

On peut de cette manière remplacer un arc de routage nomscript --> nomNoeud en introduisant directement la modification dans le nœud.

Le fonctionnement du routage peut-être vu comme l'activation automatique d'un script prédéfini assurant le parcours du graphe à partir de l'insatant initial, avec, pour chaque arc, copie de la valeur duchamp d'origine dans le champ d'arrivée.

- Accéder dans le script à un paramètre 'ordinaire' (valeurs simples, vecteurs) du prototype

```
field SFFloat   hautC_scr IS  hautC
```

syntaxe x3d

```
<field     name="rayonB_scr"     type="SFFloat"
value="0.0" accessType="initializeOnly"/>
<IS>
<connect      nodeField="rayonB_scr"
              protoField="rayonB"/>
</IS>
```

- Accéder dans le script à un paramètre événement du prototype

Param prototype :

```
eventIn SFTime   debut
```

Param script :

```
eventIn SFTime debut_scr IS debut
```

syntaxe x3d

Param prototype :

```
<ProtoDeclare name="PION">
    <ProtoInterface>
      <field  name="debut" type="SFTime"
            accessType="inputOnly"/>
```

Param script :

```
<Script DEF="SCR">
<field name="debut_scr" type="SFTime"
 accessType="inputOnly"/>
```

remarquer l'absence de connexion explicite avec "debut"

Exemple animation non réalisable sans script changement de valeur de noeud

Variante de l'exemple précédent : changer la valeur du champ choice directement dans le script, sans routage script-switch

- Les fonctions du script sont accessibles par routage
- Une fonction du script peut accéder à la représentation interne d'un noeud nommé et modifier le champ choice
- L'interface du script doit lier le nom VRML du noeud à une référence sur la représentation mémoire de ce noeud, au travers d'un paramètre du script

```
#VRML V2.0 utf8
#feuxalternScript.wrl vert/jaune alternés
#le script change directement le contenu du
noeud SWN
Transform {
    children [
      DEF TOS TouchSensor {}
      DEF SWN Switch {
        whichChoice 0
        choice [
          Shape {
            geometry DEF C Cone {}
            appearance Appearance {
            material Material {
                  diffuseColor .3 .3 1 }}}
          Shape {
            geometry USE C
            appearance Appearance {
            material Material {
                diffuseColor 1 1 0 }}}]
```

```
      }
         DEF SPRG2 Script {
         eventIn SFTime instantClic
         field SFBool couleurUn FALSE
         field SFNode SWN_scr USE SWN
         url "javascript:
           function initialize() {
             directOutput = TRUE ;
           }
    function instantClic( value, time )
{     couleurUn = !couleurUn;
       if (couleurUn) {
          SWN_scr.whichChoice = 1; }
       else { SWN_scr.whichChoice = 0; }
          }"
       }
    ]
  }

ROUTE TOS.touchTime TO SPRG2.instantClic
```

Version X3D

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ...   >
<X3D profile="Full">
  <head>
    <meta name="filename"
 content="FeuxAlternScript.x3d"/>
    <meta name="generator"
content="Vrml97ToX3dNist,
http://ovrt.nist.gov/v2_x3d.html"/>
  </head>
  <Scene>
  <Transform >
       <TouchSensor  DEF="TOS" />
       <Switch  DEF="SWN"  whichChoice="0">
         <Shape >
             <Appearance >
                 <Material
                  diffuseColor="0.3 0.3 1.0"/>
             </Appearance>
                <Cone  DEF="C" />
         </Shape>
         <Shape >
             <Appearance >
                 <Material
                  diffuseColor="1.0 1.0 0.0"/>
             </Appearance>
                <Cone  USE="C"/>
           </Shape>
       </Switch>
```

```
        <Script DEF="SPRG2">
          <field name="couleurUn"
               type="SFBool" value="false"
               accessType="initializeOnly"/>
          <field name="instantClic"
               type="SFTime"
               accessType="inputOnly"/>
          <field name="SWN_scr"
            type="SFNode"
            accessType="initializeOnly">
            <Switch USE="SWN"/>
          </field>
          <![CDATA[javascript:
            function initialize() {
              directOutput = TRUE ;
            }

            function instantClic( value, time ) {
               couleurUn = !couleurUn;
               if (couleurUn)
                  SWN_scr.whichChoice = 1;
               else SWN_scr.whichChoice = 0;
            }
          ]]>
        </Script>
    </Transform>
<ROUTE fromNode="TOS"
       fromField="touchTime" toNode="SPRG2"
       toField="instantClic"/>
    </Scene>
</X3D>
```

Exemple 6:

Feu alterné *sans routage entre script et le noeud switch* : le script change directement le contenu du noeud SWN.

Initialisation de donnée du script à partir dù n nœud nommé par DEF dans le fichier VRML :

`field SFNode SWN_scr USE  SWN`

Modification des champs dans le code (lecture et modification)

`SWN_scr.whichChoice = 1;`

Le switch SWN des exemples 4-5 est inchangé

```
DEF SPRG2 Script {
    eventIn SFTime instantClic
    field SFBool couleurUn FALSE
    field SFNode SWN_scr USE  SWN
    url "javascript:
        function initialize() {
            directOutput = TRUE ;
        }
        function instantClic( value, time ) {
        // permute l'état sélectionnant couleur
            couleurUn = !couleurUn;
            if (couleurUn)
                SWN_scr.whichChoice = 1;
            else SWN_scr.whichChoice = 0;
            }"
    }
……
#un seul arc de routage pour activer le script
ROUTE TOS.touchTime TO SPRG2.instantClic
```

Script et enrichissement de la scène
Création/destruction de nœuds

- on peut créer de nouveaux objets avec Javascript.
- on peut modifier des nœuds existant, particulièrement la liste des fils (attribut `children` d'un nœud `Transform`)
- toute la hiérarchie est finalement accessible au programmeur de script au travers du DOM !

Constructeurs de `SFNode` *et* `MFNode`

`SFNode` à partir d'une chaine de caractères :

```
new SFNode( 'Shape{geometry Box{size 0.5 0.5 0.5 }}')
```

.q. ui peut inclure des variables, cote le flottant `cote` :

```
nœud =
new SFNode('Shape{geometry Box{size ' + cote +
' '+ cote +' '+ cote + ' }}')
```

`MFNode` : nombre variable de paramètres de type `SFNode`.

```
tabN = new MFNode(nœud) ;
  tt = new MFNode() //crée un tableau vide.
```

On ajoute des éléments :

```
tt[0] = …
tabn[1] = new SFNode(
   'Shape{geometry Sphere {radius 1.3 }}');
```

Exemple 7

Création dù n nœud lors de lîn itialisation, disparition par clic

```
Transform{
    children [
        DEF FormeVideN Shape {}
    ]
 }

Transform {
    translation -3 -3 0
    children[
        DEF ToS TouchSensor {}
        Shape {geometry Sphere{}}
        DEF AjoutCubeP Script {
            field  SFBool directOutput TRUE
            field  SFNode FormeVideN_scr       USE
FormeVideN
        url "javascript:
            var cube; //globale
            function initialize(){
            cube = new
                SFNode('Box{size 0.5 0.5 0.5 }');
                formevide.geometry = cube;
            }
            function changerTime(){
            if (formevide.geometry )
                formevide.geometry = null;
            else formevide.geometry = cube;"
        }
    ]
}
ROUTE ToS.touchTime TO AjoutCubeP.changerTime
          SFNode('Box{size 0.5 0.5 0.5 }'); }"
```

Des événements de type MFNode ...

Transform contient deux événements entrants **addchildren** et **removeChildren** de type **MFNode**

Exemple 8 :

Construire une diagonale de petits cubes dans TransfN
Ajoute un cube au tableau children à chaque appui sur la boule

```
#le noeud est initialement vide
DEF TransfN Transform {
        translation -4 -4 0   #bas gauche
        scale 0.5 0.5 0.5
        children[ ]        #tableau à remplir
}

Transform {
    translation -3 3 0
    children [
        DEF BLOC Shape {geometry Sphere{radius 0.5}}
        #boule: l'appui crée un cube (voir ROUTE)
        DEF ToS TouchSensor{}
        DEF AjoutCubeP Script {
            eventIn SFTime changerTime
            eventOut MFNode nouvCubes
            # ajouter un petit cube à chaque activation
            field SFVec3f translat 0 0 0
            field SFVec3f incrTranslat 0.7 0.7 0
            field SFNode cube NULL # un seul objet à
                                   # partager
            url "javascript:
                function initialize(){
                    //une référence de cube
                    cube = new SFNode(
        'Shape{geometry Box{size 0.5 0.5 0.5 }}');
                }
```

```
        function changerTime( value, time ) {
            nouvTransform = new SFNode('Transform{}');
            translat = translat.add(incrTranslat);
            nouvTransform.translation = translat;
            nouvTransform.children[0]= cube ;// == USE
            //en faire un tableau de noeuds
            nouvCubes = new MFNode(nouvTransform);
         }
    "}
  ]
}

ROUTE ToS.touchTime TO AjoutCubeP.changerTime
ROUTE AjoutCubeP.nouvCubes TO TransfN.addChildren
```

VARIANTE SANS ROUTAGE VERS **TRANSFN**

On manipule directement le noeud **TransfN** dans la fonction changerTime du script, en ajoutant un élément à son tableau `children` ; le noeud nommé à l'extérieur doit être associé explicitement dans l'entête du script.

```
 DEF AjoutCubeDirectP Script {
    eventIn SFTime changerTime
    field SFNode transfN_scr USE TransfN
    #pour ajouter un cube a TransfN
    field SFVec3f translat 0 0 0
    field SFVec3f incrTranslat 0.7 0.7 0
    field SFNode cube NULL
    field SFInt32 k 0#compte les fils de TransfN
    url "javascript:
      function initialize(){…}
      function changerTime( value, time ) {
        …
        nouvTransform.children[0] = cube;
        // on ajoute directement au noeud TransfN
        transfN_scr.children[k++] = nouvTransform;
        }
     "
     }
   ]
 }
ROUTE ToS.touchTime TO AjoutCubeDirectP.changerTime
```

Script et prototype l'événement `initialize` pour calculer

Un prototype ne peut pas faire de calcul .

Mais un script peut modifier les champs du prototype quand on vient d'instancier : l'événement **`initialize`** et son traitement associé intervient au chargement, avant l'affichage.

- L'objet affiché sera conforme aux souhaits du concepteur.
- Le calcul est lancé indépendamment pour chaque appel du prototype avec les valeurs de ses paramètres effectifs

```
PROTO NomProto [ ] #corps
   {
    DEF nomSommetN Transform{
        children[
           #description de la hiérarchie
           …
           # le script anonyme, exécuté à
           # chaque appel du prototype
           Script {
              #nœuds relais
              url : "javascript :
                 function initialize(){
 #calculs de valeurs dépendant des paramètres,
 #modification de champs des noeuds relais

              } "   #fin javascript
   }#fin noeud script
 }#fin proto
```

Exemple 9 : Proto Pion

On veut créer des pions de différentes tailles, en donnant seulement la hauteur du cone, les rayons du cone et de la sphère ; la translation de la sphère est calculée

L'entête :

```
PROTO PION [     #déclaration des paramètres
   field SFColor    couleurC 0.5 0.5 0.5
   field SFColor    couleurB 0.8 0.8 0.8
    field SFFloat    hautC 3.0
    field SFFloat    rayonC 0.75
    field SFFloat    rayonB 0.5
 ]
```

- *Le début du corps*, nommé :

```
#corps, definition du prototype, nommer le noeud
{
    DEF TransformPourPion Transform{
    #les composants (cône et sphère) sans les
    #translations
    children [
        # children[0] : le cône
        # la transformation du cone, identité
        Transform {
            children #le cône et sa couleur
                Shape {
                appearance Appearance {
                    material Material{
                        diffuseColor IS couleurC
                    }
                }
                geometry Cone {
                    bottomRadius IS rayonC
                    height IS hautC
                }
        }
```

```
}
        #children[1] : la boule et sa
        # transformation
        Transform{
            children Shape {
                appearance Appearance {
                    material Material {
                        diffuseColor IS couleurB
                    }
                }
                geometry Sphere {
                    radius IS rayonB
                }
            }
        }
    ]
}
```

- *Le script* anonyme qui appartient au corps

```
# le calcul des translations et leur mise en place
# dans le nœud du prototype children[2]
Script {
    directOutput    TRUE
    field SFFloat   hautC_scr IS  hautC
    field SFNode    TransformPourPion_scr
            USE TransformPourPion
    url " javascript :
        function initialize(){
        //calcul des translations
        //récupérer la première transformation,
        tcone = TransformPourPion_scr.children[0] ;
        //la modifier pour poser le cône sur le
        //plan y=0
        tcone.translation =
            new SFVec3f(0, hautC_scr /2,0) ;
        //ou bien
        tcone.translation.x = 0;
```

```
            tcone.translation.z=0;
            tcone.translation.y= hautC_scr /2;

            //deux étapes en une…pour la sphère,
            //perchée au sommet du cône
            TransformPourPion_scr.
                children[1].translation =
                    new SFVec3f(0, hautC_scr ,0) ;
            }"
    }
} # fin prototype !
```

- *Utilisation* du prototype avec calcul initial

```
# Création de la scène avec quelques pions :
Transform {
    children [
        PION {} ,
        Transform {
            translation 2 0 0
            children PION {
                couleurC 0 0 1
                couleurB 1 1 0
                hautC 6
                rayonC 1
                rayonB 2
            }
        } ,
        Transform{
            translation -3 0 0
            children PION {
                couleurC 0 1 1  couleurB 1 0 1
                hautC 5
                rayonC 0.5
                rayonB 1.5
            }
        }
    ]
}
```

Exemple 10 : Pyramide

Le calcul permet de définir un `IndexedFaceSet` à partir de quelques paramètres. C'est ainsi que `Box`, `Cone`, `Sphere`, `Cylindre` sont définis, prototype à scripts prédéfinis.

Il manque `Pyramide` :

- Une pyramide à base carrée dans le plan y=0, centrée à l'origine, de coté **`cote`** de hauteur **`haut`** de sommet-pointe (numéro 4) sur l'axe y, coord (0 `haut` 0)
- Schéma des sommets vus du dessus (0, n*haut , 0) :

```
3         0
     4
2         1
```

- Les sommets de la base ont pour coordonnée x et z ± cote/2

```
PROTO Pyramide [
    field MFColor couleursFaces
  [1 0 0, 1 0 0.5 , 1 1 1, 1 1 0, 0.5 0.5 0.5]
    field SFFloat haut 1
    field SFFloat cote 2
]{
    DEF pyram Shape {
        geometry IndexedFaceSet {
             color DEF Couleurs  Color {
                  color [1 0 0, 1 0 0.5 , 1 1 1,
                         1 1 0, 0.5 0.5 0.5]
               }
             coord DEF CoordPoints Coordinate {
                  point [ 1 0 1, 1 0 -1,
                         -1 0 -1, -1 0 1, 0 3 0]
               }
            coordIndex [ 0 1 4 -1  1 2 4 -1
                         2 3 4 -1  3 0 4 -1
                         0 3 2 1 -1]
     colorPerVertex FALSE}
  }
```

```
  Script{
     directOutput TRUE
     #les variables relais des paramètres
     field MFColor couleursProto_scr IS couleursFaces
     field SFFloat    haut_scr IS haut
     field SFFloat    cote_scr IS cote
     #les variables relais des noeuds nommés
     field SFNode CoordPoints_scr USE CoordPoints
     field SFNode  Couleurs_src USE Couleurs
     url "javascript:
         function initialize(){
         //paramètre diffère de la valeur par défaut
           if (cote_scr != 2){
           //coordonnées des quatre coins(y=0)
            xz=cote_scr / 2;
            CoordPoints_scr.point[0].x=xz;
            CoordPoints_scr.point[0].z=xz;
            CoordPoints_scr.point[1].x=xz;
            CoordPoints_scr.point[1].z=-xz;
            CoordPoints_scr.point[2].x=-xz;
            CoordPoints_scr.point[2].z=-xz;
            CoordPoints_scr.point[3].x=-xz;
            CoordPoints_scr.point[3].z=xz;
         }
         //idem haut differe defaut
         if (haut_scr != 1)
             CoordPoints_scr.point[4].y=haut_scr;
         //on recopie les couleurs, défaut ou pas
         for (i=0;i<5;i++)
             Couleurs_src.color[i]=
                        couleursProto_scr[i];
         }
     "
   }
}
```

Animation locale à un prototype

- *Routage interne au prototype*

Les sensors et le script portent *des noms locaux* inconnus à l'extérieur du prototype, *le routage local doit être décrit à l'intérieur du prototype*.
On veut que le rayon de la tête (un flottant) d'une instance de pion puisse être modifié après la création de l'objet.
Pour cela, on utilise un `TimeSensor` couplé par routage à une fonction du script changeant le champ rayon de la boule du pion (à la place d'interpolateur `ScalarInterpolator`).

```
PROTO []{
# hiérarchie contenant
# TimeSensor DEF TIS
}
DEF SCR Script {

  }#fin script
  ROUTE SCR.demarre_tis TO TIS.startTime
  ROUTE TIS.fraction_changed TO SCR.fraction
}#fin proto
```

- ***Du routage externe au routage interne***

...Mais le démarrage de l'animation doit se faire à partir d'un événement extérieur, typiquement un clic sur un **`TouchSensor`**.

La solution est :

- d'ajouter un **`eventIn`** au prototype (qui sera propre à chaque instance au même titre que les autres paramètres)
- d'ajouter un **`eventIn`** au script qui sera lié au paramètre du prototype au travers du mot clé IS, comme pour les paramètres valeur.

Paramètre du protoype :

```
eventIn SFTime   debut
```

Paramètre eventIn du script(fonction associée)

```
eventIn SFTime debut_scr IS debut
```

En supposant qu'il existe plusieurs instances nommées du script:

```
Transform{translation -6 0 0
     children DEF Pion4 PION{couleurC 1 1 1
couleurB 1 0 0 hautC 2 rayonC 1 rayonB 1.5}}
]}
```

routage au niveau global :

```
ROUTE TOS.touchTime TO Pion2.debut
ROUTE TOS.touchTime TO Pion4.debut
```

Un événement touchTime externe, transmis à Pion2.debut est transmis vers debut_scr qui lui-même démarre le timesensor local de l'instance; ensuite, le routage interne appelle de manière 'continue' la fonction fraction du script qui modifie le rayon de la boule.

Exemple 11

On veut modifier le rayon d'une sphère dans une instance de prototype après la création de l'objet.

Plus précisément, le rayon de la tête de certains pions doit varier, en l'associant à un TimeSensor (durée 5). Les horloges des pions sont activés par un TouchSensor.

Un nouvelle fonction du script reçoit l'événement fraction_changed et va modifier le changement d'échelle de la sphère de chaque instanciation du prototype

```
PROTO PION [
   . . . #ne change pas
    ]
    #corps, définition du prototype
    {
        DEF TransformPourPion Transform {
            . . .
            #le cône et la sphère
        }
        # le TimeSensor délivre des valeurs
        # entre 0 et 1
        DEF TIS TimeSensor {
            cycleInterval 5
            enabled FALSE
            loop TRUE
        }
   }
```

...

```
DEF ChangerBouleP Script {
    directOutput TRUE
    eventIn SFFloat    fraction
    eventOut SFBool    enabled
    field SFFloat      hautC_scr IS  hautC
    field SFNode       TransformPourPion_scr  USE
                       TransformPourPion
    field SFFloat      rayonB_scr IS rayonB
    url "javascript:
        function initialize(){. . .}

        function fraction(value) {
            rayon = rayonB_scr * (1 - 0.9 * value);
                TransformPourPion_scr.
                    children[2].scale[0]=rayon;
                TransformPourPion_scr.
                    children[2].scale[1]=rayon;
                TransformPourPion_scr.
                    children[2].scale[2]=rayon;
        }"
    }#fin script

    ROUTE ChangerBouleP.enabled TO TIS.enabled
    ROUTE TIS.fraction TO ChangerBouleP.fraction
}#fin proto
```

Exemple 12

Un noeud Text de la hiérarchie affiche le nombre de cycles complets effectués par un timeSensor de durée 1 (0 initialement, modifié par script ensuite) ; le script arrête l'animation après 7 cycles complets et salue l'utilisateur.

```
# l'horloge
  DEF TiS TimeSensor {
      cycleInterval 1
      loop TRUE
    }

# Le noeud d'affichage
Transform {translation -1.5 1 0
   children [
    Shape {
        geometry DEF textN Text {
            //valeur initiale
            string ["nbcycles = ", "0",""]
            fontStyle FontStyle{}
            }
        appearance Appearance {
            material Material { diffuseColor 1 1 1}}
    }
    #le graphe de transmission des événements
    #le script s'appelle AffichNbCyclesP

    #--chaque début de cycle :
    ROUTE TiS.cycleTime_changed TO
                AffichNbCyclesP.set_debutCycle
    ##--lorsque le nombre max de cycles est atteint
    ROUTE AffichNbCyclesP.finBoucle_changed TO
              TiS.set_enabled
```

```
#le noeud script
DEF AffichNbCyclesP Script {
  #les paramètres, partie VRML
    directOutput       TRUE
    eventIn  SFTime    debutCycle
    eventOut SFBool    finBoucle
    field    SFNode    txt USE textN
    url  "javascript:
        //global et conservé entre les appels
        var nbCycles=0;
        function debutCycle(value,timeStamp){
            //appel chaque seconde,
            // début de cycle TimeSensor
            if (nbCycles == 0)
                //active le TimeSensor
                finBoucle = true;
            nbCycles ++;
            if (nbCycles == 7) {
                //arrête le TimeSensor
                finBoucle=false;
                //affichage final
                txt.string[2]='C\'est fini,au revoir!';
            } else
                //modification affichage compteur
                txt.string[1] = nbCycles;
        }"
}#fin script

]
}
```

Traduction protopyramide.x3d

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
"http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D profile="Full">
  <head>
    <meta name="filename" content="protopyramide.x3d"/>
    <meta name="generator" content="Vrml97ToX3dNist,
http://ovrt.nist.gov/v2_x3d.html"/>
  </head>
  <Scene>
  <ProtoDeclare name="Pyramide">
    <ProtoInterface>
      <field  name="haut" type="SFFloat" value="1.0"
accessType="initializeOnly"/>
      <field  name="cote" type="SFFloat" value="2.0"
accessType="initializeOnly"/>
      <field  name="couleursFaces" type="MFColor" value="1.0 0.0 0.0,
1.0 0.0 0.5, 1.0 1.0 1.0, 1.0 1.0 0.0, 0.5 0.5 0.5, "
accessType="initializeOnly"/>
    </ProtoInterface>
    <ProtoBody>
          <Shape  DEF="pyram" >
              <IndexedFaceSet  coordIndex=" 0 1 4 -1 1 2 4 -1 2 3 4 -1 3
0 4 -1 0 3 2 1 -1" colorPerVertex="false">
                  <Coordinate  DEF="CoordPoints"  point="1.0 0.0 1.0,
1.0 0.0 -1.0, -1.0 0.0 -1.0, -1.0 0.0 1.0, 0.0 3.0 0.0, "/>
                  <Color  DEF="Couleurs"  color="1.0 0.0 0.0, 1.0 0.0
0.5, 1.0 1.0 1.0, 1.0 1.0 0.0, 0.5 0.5 0.5, "/>
              </IndexedFaceSet>
          </Shape>
          <Script  directOutput="true">
            <field name="CoordPoints_scr" type="SFNode"
accessType="initializeOnly">
              <Coordinate USE="CoordPoints"/>
            </field>
            <field name="haut_scr" type="SFFloat" value="0.0"
accessType="initializeOnly"/>
            <field name="cote_scr" type="SFFloat" value="0.0"
accessType="initializeOnly"/>
            <field name="Couleurs_src" type="SFNode"
accessType="initializeOnly">
              <Color USE="Couleurs"/>
            </field>
            <field name="couleursProto_scr" type="MFColor" value=""
accessType="initializeOnly"/>
            <IS>
            <connect nodeField="haut_scr" protoField="haut"/>
            <connect nodeField="cote_scr" protoField="cote"/>
            <connect nodeField="couleursProto_scr"
protoField="couleursFaces"/>
```

```
</IS>
```

```
            <![CDATA[javascript:
function initialize(){

//On regarde si l'un des paramètres diffère de la valeur par défaut

if (cote_scr != 2)
{// changer les coordonnees des quatre coins +-demi-coté en x et z
//y ne change pas, 0
xz=cote_scr / 2;
CoordPoints_scr.point[0].x=xz;CoordPoints_scr.point[0].z=xz;
CoordPoints_scr.point[1].x=xz;CoordPoints_scr.point[1].z=-xz;
CoordPoints_scr.point[2].x=-xz;CoordPoints_scr.point[2].z=-xz;
CoordPoints_scr.point[3].x=-xz;CoordPoints_scr.point[3].z=xz;}
if (haut_scr != 1)
CoordPoints_scr.point[4].y=haut_scr;
//on recopie les couleurs, défaut ou pas
for (i=0;i<5;i++)
{Couleurs_src.color[i]=couleursProto_scr[i];}
}
            ]]>
          </Script>
    </ProtoBody>
  </ProtoDeclare>
  <Transform >
      <ProtoInstance name="Pyramide">
        <fieldValue name="haut" value="2.0"/>
        <fieldValue name="cote" value="3.0"/>
      </ProtoInstance>
      <Transform  translation="3.0 0.0 0.0">
          <ProtoInstance name="Pyramide">
            <fieldValue name="haut" value="2.0"/>
            <fieldValue name="cote" value="1.0"/>
            <fieldValue name="couleursFaces" value="0.0 1.0 0.0, 0.0 1.0
1.0, 0.0 0.0 1.0, 1.0 1.0 0.0, 1.0 0.0 0.0, "/>
          </ProtoInstance>
      </Transform>
  </Transform>
  <Viewpoint  position="20.0 20.0 20.0" orientation="-1.0 1.0 0.0
0.785" description="20 20 20"/>
  <Viewpoint  position="3.0 7.0 0.0" orientation="1.0 0.0 0.0  -1.57"
description="au dessus x=3"/>
  <Viewpoint  position="0.0 5.0 0.0" orientation="1.0 0.0 0.0  -1.57"
description="au dessus x=0"/>
  <Viewpoint  position="1.5 -7.0 0.0" orientation="1.0 0.0 0.0  1.57"
description="au dessous x=1.5"/>
  <Viewpoint  position="10.0 0.0 0.0" orientation="0.0 1.0 0.0  1.57"
description="droite"/>
  </Scene>

</X3D>
```

Traduction protoScrPionTimExt.x3d

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
"http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D profile="Full">
  <head>
    <meta name="filename" content="protoScrPionTimExt.x3d"/>
    <meta name="generator" content="Vrml97ToX3dNist,
http://ovrt.nist.gov/v2_x3d.html"/>
  </head>
  <Scene>
  <ProtoDeclare name="PION">
    <ProtoInterface>
      <field  name="debut" type="SFTime" accessType="inputOnly"/>
      <field  name="couleurC" type="SFColor" value="0.5 0.5 0.5"
               accessType="initializeOnly"/>
      <field  name="couleurB" type="SFColor" value="0.8 0.8 0.8"
               accessType="initializeOnly"/>
      <field  name="rayonC" type="SFFloat" value="0.75"
                accessType="initializeOnly"/>
      <field  name="rayonB" type="SFFloat" value="0.5"
                accessType="initializeOnly"/>
      <field  name="hautC" type="SFFloat" value="3.0"
               accessType="initializeOnly"/>
    </ProtoInterface>
    <ProtoBody>
          <Transform  DEF="TransformPourPion" >
              <TimeSensor  DEF="TIS"  stopTime="-1.0" loop="false"
                           enabled="true" cycleInterval="3.0"/>
              <Transform >
                  <Shape >
                      <Appearance >
                          <Material >
                              <IS>
               <connect nodeField="diffuseColor" protoField="couleurC"/>
                              </IS>
                          </Material>
                      </Appearance>
                      <Cone >
                          <IS>
                <connect nodeField="height" protoField="hautC"/>
                <connect nodeField="bottomRadius" protoField="rayonC"/>
                          </IS>
                      </Cone>
                  </Shape>
              </Transform>
              <Transform >
                  <Shape >
                      <Appearance >
                          <Material >
                              <IS>
```

```
            <connect nodeField="diffuseColor" protoField="couleurB"/>
                            </IS>
                        </Material>
                    </Appearance>
                    <Sphere >
                        <IS>
             <connect nodeField="radius" protoField="rayonB"/>
                        </IS>
                    </Sphere>
                </Shape>
              </Transform>
          </Transform>

          <Script DEF="SCR">
        <field name="TransformPourPion_scr" type="SFNode"
                     accessType="initializeOnly">
              <Transform USE="TransformPourPion"/>
            </field>
        <field name="debut_scr" type="SFTime" accessType="inputOnly"/>
        <field name="demarre_tis" type="SFTime" accessType="outputOnly"/>
        <field name="hautC_scr" type="SFFloat" value="0.0"
                      accessType="initializeOnly"/>
       <field name="fraction" type="SFFloat" accessType="inputOnly"/>
            <IS>
            <connect nodeField="debut_scr" protoField="debut"/>
            <connect nodeField="hautC_scr" protoField="hautC"/>
            </IS>
            <![CDATA[javascript:
function initialize(){
     tcone = TransformPourPion_scr.children[1];
//et definir la translation posant le cone sur y=0
     tcone.translation.x = 0;tcone.translation.z=0;
     tcone.translation.y= hautC_scr /2;
//TransformPourPion_scr est un nom relatif : designe l'instance activee
// evt initial provenant de l'exterieur, sur P2 ou sur P4
     tboule = TransformPourPion_scr.children[2].translation = new
SFVec3f(0, hautC_scr ,0);
     }

//nouvelle fonction pour initialiser le TimeSensor
     function debut_scr(value){demarre_tis = value;}

//variation avec le temps
     function fraction(value){
     echelle =  (1 - 0.9 * value);
TransformPourPion_scr.children[2].scale[0]=echelle;
TransformPourPion_scr.children[2].scale[1]=echelle;
TransformPourPion_scr.children[2].scale[2]=echelle;
     }
            ]]>
          </Script>
```

```
<ROUTE fromNode="SCR" fromField="demarre_tis" toNode="TIS"
toField="startTime"/>
<ROUTE fromNode="TIS" fromField="fraction_changed" toNode="SCR"
toField="fraction"/>
    </ProtoBody>
  </ProtoDeclare>


  <Transform >
      <TouchSensor  DEF="TOS" />
      <ProtoInstance name="PION"> #par défaut
      </ProtoInstance>
      <Transform  translation="2.0 0.0 0.0">
          <ProtoInstance name="PION" DEF="Pion2">
            <fieldValue name="couleurC" value="0.0 0.0 1.0"/>
            <fieldValue name="couleurB" value="1.0 1.0 0.0"/>
            <fieldValue name="rayonC" value="1.0"/>
            <fieldValue name="rayonB" value="2.0"/>
            <fieldValue name="hautC" value="6.0"/>
          </ProtoInstance>
      </Transform>
      <Transform  translation="-3.0 0.0 0.0">
          <ProtoInstance name="PION">
            <fieldValue name="couleurC" value="0.0 1.0 1.0"/>
            <fieldValue name="couleurB" value="1.0 0.0 1.0"/>
            <fieldValue name="rayonC" value="0.5"/>
            <fieldValue name="rayonB" value="1.5"/>
            <fieldValue name="hautC" value="5.0"/>
          </ProtoInstance>
      </Transform>
      <Transform  translation="-6.0 0.0 0.0">
          <ProtoInstance name="PION" DEF="Pion4">
            <fieldValue name="couleurC" value="1.0 1.0 1.0"/>
            <fieldValue name="couleurB" value="1.0 0.0 0.0"/>
            <fieldValue name="rayonC" value="1.0"/>
            <fieldValue name="rayonB" value="1.5"/>
            <fieldValue name="hautC" value="2.0"/>
          </ProtoInstance>
      </Transform>
  </Transform>
<ROUTE fromNode="TOS" fromField="touchTime" toNode="Pion2"
toField="debut"/>
<ROUTE fromNode="TOS" fromField="touchTime" toNode="Pion4"
toField="debut"/>
  <Viewpoint  position="10.0 10.0 10.0" orientation="-1.0 1.0 0.0
0.785" description="10 10 10"/>
  </Scene>

</X3D>
```