# Informatique Graphique : TP5

## Partie 1 : Interpolation Framework

Pour interpoler correctement la position d'un objet, il faut connaître sa position à différentes Keyframes. Une fois que l'on connait ces Keyframes, et à quel temps elles doivent être effectives, on calcule le facteur d'interpolation, qui correspond simplement à un coefficient indiquant si on est plus proche de la prochaine Keyframe que de la précédente.

```
std::array< Keyframe, 2 > result = getBoundingKeyframes( time );
float factor =  (time - result[0].first) / (result[1].first - result[0].first);
```

On applique ensuite les transformations en faisant une interpolation linéaire, l'interpolation est faite avec GLM grâce aux méthodes `lerp` et `slerp`.

```
glm::quat orientation = glm::slerp(result[0].second.getOrientation(),
      result[1].second.getOrientation(), factor);
glm::vec3 translation = glm::lerp(result[0].second.getTranslation(),
      result[1].second.getTranslation(), factor);
glm::vec3 scale = glm::lerp(result[0].second.getScale(),
      result[1].second.getScale(), factor);
```

Enfin on renvoie la matrice des transformations.

```
return GeometricTransformation( translation, orientation, scale ).toMatrix();
```

Si le temps est avant la première Keyframe, ou après la dernière, on renvoie à la place la première ou dernière Keyframe.

```
if( time <= itFirstFrame->first ) return itFirstFrame->second.toMatrix();
if( time >= itLastFrame->first ) return itLastFrame->second.toMatrix();
```

Enfin dans le cas où l'objet n'a aucune Keyframes de définie, on renvoie la matrice identité.

## Partie 2 : Application avec un objet

Pour cette application, on va utiliser un cylindre que l'on fera tourner et se déplacer. On commence par créer le cylindre à la manière de n'importe quel Renderable.

```
auto cylinder = std::make_shared<KeyframedCylinderRenderable>(flatShader);
```

On définit ensuite un certain nombre de transformations que l'on souhaite appliquer au cylindre.

```
GeometricTransformation none;
GeometricTransformation r1(glm::vec3(0, 0, 0), glm::rotate(glm::mat4(1.0),
      glm::radians(180.0f), glm::vec3(0, 0, 1)));
GeometricTransformation r2(glm::vec3(0, 0, 0), glm::rotate(glm::mat4(1.0),
      glm::radians(360.0f), glm::vec3(0, 0, 1)));
```

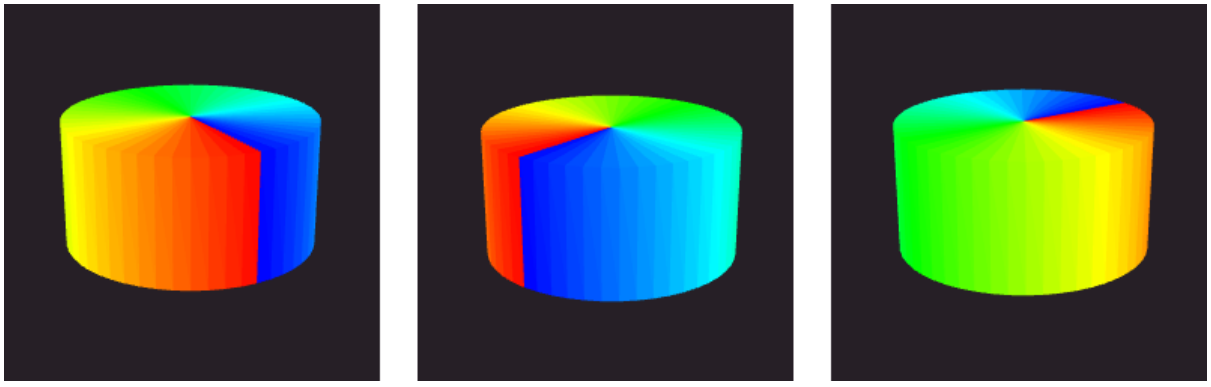
On ajoute ensuite un certain nombre de Keyframes à ce cylindre.

```
cylinder->addLocalTransformKeyframe(none, 0.0f);
cylinder->addLocalTransformKeyframe (r1, 1.5f);
cylinder->addLocalTransformKeyframe (r2, 3.0f);
```

De la même manière que les `HierarchicalRenderable` (dont la classe hérite), les `parentTransformKeyframe` correspondent à une transformation par rapport au parent alors que les `localTransformKeyframe` correspondent à une transformation locale (ici il n'y a pas vraiment de différence puisqu'on n'a qu'un objet).

On lance ensuite l'animation dans le `Viewer` et on définit qu'elle boucle toutes les trois secondes puisque c'est le temps de notre animation.

```
viewer.startAnimation();
viewer.setAnimationLoop(true, 1.0);
```



*Résutat obtenu avec le code précédent, le cylindre tourne sur lui-même*

## Partie 3 : Application avec hiérarchie

On commence par créer la racine de notre hiérarchie puis son fils de la même manière que n'importe quel Renderable. On définit ensuite le fils comme étant décalé de deux unités par rapport au père selon l'axe Y grâce à la méthode `setParentTransform()`.

```
KeyframedCylinderRenderablePtr root =
	std::make_shared< KeyframedCylinderRenderable>(flatShader);
KeyframedCylinderRenderablePtr c1 =
	std::make_shared< KeyframedCylinderRenderable>(flatShader);
c1->setParentTransform(
	glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 2.0f, 0.0f)));
```

On définit ensuite les transformations qui seront appliquées à la racine et on les lui ajoute, ici on fera une animation de « yoyo » vertical.

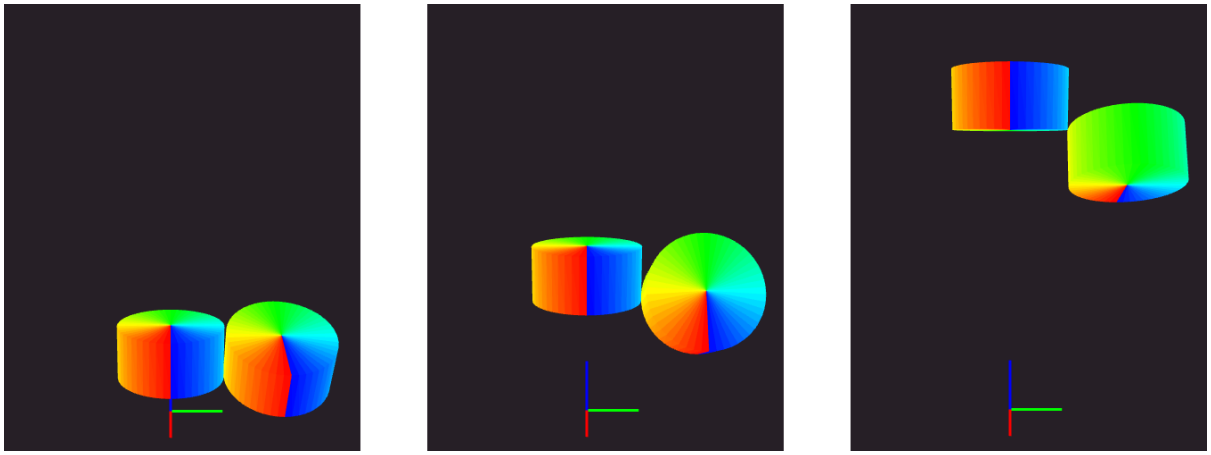
```
GeometricTransformation none;
GeometricTransformation t1(glm::vec3(0, 0, 6.0f));

root->addParentTransformKeyframe(none, 0.0f);
root->addParentTransformKeyframe(t1, 1.5f);
root->addParentTransformKeyframe(none, 3.0f);
```

Même chose pour le fils, on va cette fois appliquer uniquement des transformations locales puisqu'on souhaite conserve le décalage de deux unités définit plus haut. On définit ici une animation de rotation selon l'axe Y.

```
GeometricTransformation r1(glm::vec3(0, 0, 0), glm::rotate(glm::mat4(1.0),
	glm::radians(180.0f), glm::vec3(0, 1, 0)));
GeometricTransformation r2(glm::vec3(0, 0, 0), glm::rotate(glm::mat4(1.0),
	glm::radians(360.0f), glm::vec3(0, 1, 0)));
c1->addLocalTransformKeyframe(none, 0.0f);
c1->addLocalTransformKeyframe(r1, 1.5f);
c1->addLocalTransformKeyframe(r2, 3.0f);
```

Enfin on définit la hiérarchie avec la méthode `HierarchicalRenderable::addChild()`, on ajoute la racine au `Viewer` et on lance l'animation et la fait boucler après trois secondes.

```
HierarchicalRenderable::addChild(root, c1);
viewer.addRenderable(root);
viewer.startAnimation();
viewer.setAnimationLoop(true, 3.0);
```



*Résultat obtenu avec le code précédent, le cylindre racine (au centre) se déplace de haut en bas, le cylindre fils suit le cylindre racine et en plus de ça tourne autour de lui-même.*