OpenGL Skeletal Animation With ASSIMP Tutorial part 5 — GameObject class

The last but not least part of this tutorial – **GameObject class**. We have already learned how to import an animated model into the OpenGL program using ASSIMP library, how to drive it's animation and render every step to the screen. But because the data is scattered all over the program it is really hard to make everything work smoothly. What we really need is a user-friendly class to load, animate and render the model easily, all in a same place... In the **GameObject class**.

Here is a declaration:

```
class GameObject
{
   private:
      ModelLoader* modelLoader; //class to import models from files

      vector < Mesh* > meshes; //here we keep meshes data
      Skeleton *skeleton; //and a skeleton

      mat4 localTransform;

   public:
      GameObject();

      void createGraphicsObject(string path); //method to load the model and create a graphical object

      void playAnimation(Animation* anim, bool reset = true); //start the animation
      void stopAnimation(); //stop the animation

      //here we can adjust the model. Often loaded models are strangely rotated or scaled, we can manualy
tweak them here
      void applyLocalRotation(float angle, vec3 axis);
      //void setLocalScale(vec3 sc);
      //void setLocalPosition(vec3 pos);

      void render(ShaderLoader* shader); //render the model to the shader

      ~GameObject();
};
```

GameObject:

-modelLoader (this is our **ModelLoader class** from the previous tutorial. It is used to import the model into the program)

-meshes vector (in this vector we will store loaded meshes by calling getModelData() function in the ModelLoader class)

-skeleton (here we will store the loaded and constructed skeleton by calling getModelData() function)

-localTransform matrix (we need this matrix to manually tweak the model. Usually imported animated models from .fbx or .dae files are strangely rotated or scaled and in order to return their normal appearance, we have to manually apply the transformation matrix to the vertices in the vertex shader)

-createGraphicsObject() (in this method we import the model and store it's information into the local storage)

-playAnimation() (this method is just a bridge between the user and the **Skeleton class** function, the names in both functions are the same)

-stopAnimation() (this one is also just a bridge between the user and the **Skeleton class** function which have the same names)

-applyLocalRotation() (this method is needed to tweak the localTransform matrix. Our program supports only rotation tweaks on the models but scaling and position functions are really similar to the one we have and writing them shouldn't cause any problems)

-render() (this function renders the model to the given shader)

Class's realization:

```
GameObject::GameObject()
{
   modelLoader = new ModelLoader();

   localTransform = mat4(1.0);
}
```

The constructor is simple. Here we define the ModelLoader class and set the localTransform matrix as a unit matrix.

```
void GameObject::createGraphicsObject(string path)
{
   modelLoader->loadModel(path); //load the model from the file
```

```
    modelLoader->getModelData(skeleton, meshes); //get the loaded data and store it in this class
}
```

In the createGraphicsObject() function we import the model from the file into our data structure kept in the local ModelLoader class. Then we call getModelData() function to extract this information into the local meshes vector and skeleton object.

```
void GameObject::playAnimation(Animation* anim, bool reset)
{
    skeleton->playAnimation(anim, reset); //play animation
}
```

We interpret this method as a bridge. We just call skeleton's playAnimation() function with same parameters.

```
void GameObject::stopAnimation()
{
    skeleton->stopAnimation(); //stop animation
}
```

Another method that we use as a bridge between the user and a Skeleton class.

```
void GameObject::applyLocalRotation(float angle, vec3 axis)
{
    vec3 sc;
    quat rot;
    vec3 tran;
    vec3 skew;
    vec4 perspective;

    decompose(localTransform, sc, rot, tran, skew, perspective);

    localTransform = mat4(1.0);
    localTransform *= translate(tran);
    localTransform *= scale(sc);
    localTransform *= rotate(radians(angle), axis) * mat4_cast(conjugate(rot));
}
```

This function is used to apply rotation to the localTransform matrix. Firstly, we define a bunch of variables is order to decompose the matrix into them and after decomposition, we construct back the localTransform matrix in the STR order.

```
void GameObject::render(ShaderLoader* shader)
{
   glUniformMatrix4fv(glGetUniformLocation(shader->ID, "localTransform"), 1, GL_FALSE,
value_ptr(localTransform));

   skeleton->update(shader); //rendering the skeleton part

   for (int i = 0; i < meshes.size(); i++) //loop through the meshes
   {
      meshes[i]->draw(shader); //rendering the mesh part
   }
}
```

This method renders the model to the given shader. Firstly, we send our localTransform matrix as a uniform variable to the shader. Secondly, we update the skeleton data (update all the keyframes and send new bones matrices to the shader). Thirdly, we render meshes to the given shader by looping through them all and calling draw() function for each mesh.

```
GameObject::~GameObject()
{
   delete modelLoader;

   for (int i = 0; i < meshes.size(); i++)
   {
      delete meshes[i];
   }

   delete skeleton;
}
```

And the last thing…

```
object = new GameObject(); //create model

object->createGraphicsObject(path("animated_models/Caterpillar/caterpillar.fbx")); //get data from file
object->applyLocalRotation(180, vec3(1, 0, 0)); //there are some problems with loading fbx files. Models
could be rotated or scaled. So we rotate it to the normal state
```

```
object->playAnimation(new Animation("MAIN", vec2(0, 245), 0.34, 10, true)); //forcing our model to play the animation (name, frames, speed, priority, loop)

object->render(objectShader); //render our model
```

This is how we create and render the model to the shader. Isn't it beautiful?

If you have done everything correctly, you should get a result like that:

This is the end of the tutorial. Hope your program works fine now, but if no – leave a comment below and I will try to help you solve the problem. Stay tuned!