**What is Xg Engine Library**

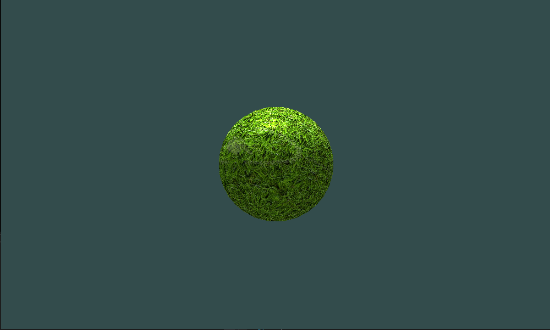
The Xg Engine Library is an object-oriented API used to create game animations and simulations. The objective of this library is to become a powerful but compact means of creating graphic assets and controlling their behavior. The Xg Engine Library defines an asset as any object that can be viewed within a scene. This could be something as static as a rock on the ground or as dynamic as a rocket seeking its target. Assets are assigned behaviors with the creation of a finite-state-machine. This mechanism allows for a systematic approach to how assets function and interact. As we will see through the following examples, developers can easily visualize their asset’s behavior, design the behavior as a finite-state-machine and implement the behavior using the Xg Engine Library.

Xg is written in C++ and works seamlessly with OpenGl. Developers do not need an OpenGl background. This is a layer hidden from the developer and accessed only through the Xg Engine Library. This layer was added purposely in order to upgrade OpenGl later for other libraries, such as Vulkan.

The example code shown demonstrates a basic **XgEngine** main program. The creation of the XgEngine object is the controlling mechanism for creating, managing and executing a Xg Engine Library scene. The **XgScene** object defines all assets, asset behaviors, light sources, and camera positions. It is the scene object that contains all pertinent objects related to the animation. Without a scene, there is nothing to animate.

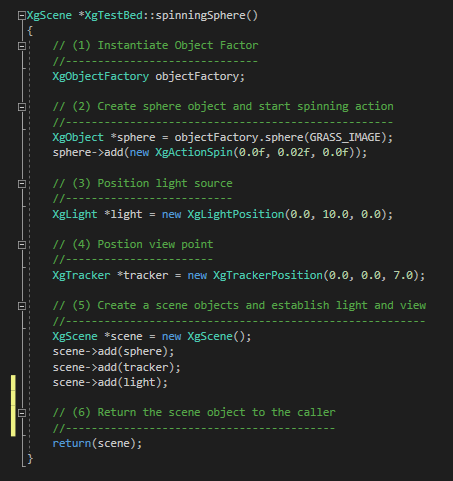
The creation and execution of the Xg Engine object is a four-step process.

1. Create the **XgEngine** object. This is done through constructors that allow control of different overall aspects of the Xg Engine. In this example, the constructor is defining the size of the animation window. But there is another constructor which will set the animation to the full screen or determine the frames-per-second. We will see this in additional examples.
2. Create the **XgScene** object. The scene object contains the animation actions and will require many addition sections to review.
3. Adding the scene to the engine. After the scene object is created, it must be added to the engine. The engine can only execute one scene at a time, adding an additional scene will override the original scene. Scenes are completely self-contained.
4. Start the animation.

**Creating a Basic Scene – A Spinning Sphere**

To illustrate the very basics of the Xg Engine we will use the API to create a spinning sphere. This is similar to the traditional “Hello World” triangle as seen in most OpenGl tutorials. In this example, a textured spinning sphere is placed in the middle of the animation, with a point light source hovering above. The position of the camera is placed on the same plane as the origin of the sphere. The light source and the camera are set at a fixed position. In future examples we will see how the light source and camera can be animated.

The creation of the spinning sphere can be accomplished in the six steps outlined in the sample code.

1. Is the instantiation of the **XgObjectFactory** object factory. The object factory provides a simple means of creating basic 3d shapes.

2. Is the creation of the **XgObject** sphere. This object is created by the object factory. A pathname is provided as the grass texture displayed. The add() function comes with two options. In this option, a “spin” action is added to the sphere. Parameters in the constructor allow for control of the spin direction and speed. In this case, the spin is applied to the Y-axis. Note, this spin action can be applied to any object.

3. Is the creation of the light source. The **XgLightPosition** creates a point light source. Later we will look at means of creating multi-lights, directional lights and spotlights. This light has no animation associated with it, since it resides at a fixed position.

4. Is the create of the camera. The **XgTrackerPosition** creates a camera position at a fixed point. The position of the camera is defined by the x, y, z coordinates in the constructor. There are other constructors that create arena, walk-through and fly-through cameras.

5. Is the creation of the scene. The **XgScene** object is created and each of the assets are added. The scene object will manage the animation of the assets. There is no predefined order that the assets must be added. If an asset is not added to the scene, it will not be animated.

6. Return the scene object.

**What is a Finite-State-Machine?**

xxx