Building up a SceneGraph in Xith3D

by Marvin Fröhlich

What is a SceneGraph?

The actual rendering is done by OpenGL. So to answer this question we first need to understand, how OpenGL works.
OpenGL renders primitives (like triangles), which are transformed by 4x4 transformation matrices. Transformations are translation, rotation and scale. Various effects like texturing or blending can be applied to these primitives. The whole scene is rendered frame by frame from a certain perspective. This perspective defines a view frustum, which is a mathematical body containing the visual part of the scene.

Of course this is not very handy for a game programmer who wants to concentrate on game logic at first place. To make life easier for a game programmer using a SceneGraph is a good way to go. You can imagine a SceneGraph like a tree built of groups containing nodes, which can be groups again or leafs. The groups can be so called TransformGroups, which indirectly hold 4x4 transformation matrices to transform all children in this group (recursively). As a consequence TransformGroups can contain other TransformGroups and the resulting transformation for a contained leaf is the product of the multiplied 4x4 transformation matrices, which is passed to OpenGL. Use one transformation per TransformGroup. The transformation of the "higher" or "outer" TransformGroup effects the "lower" or "inner" one. The leafs are instances of Shape3D in most cases. Other leaf types exist (e.g. Sound). A Shape3D describes a bunch of primitives at an abstract level together with an Appearance, which defines the effects (like texturing or blending).

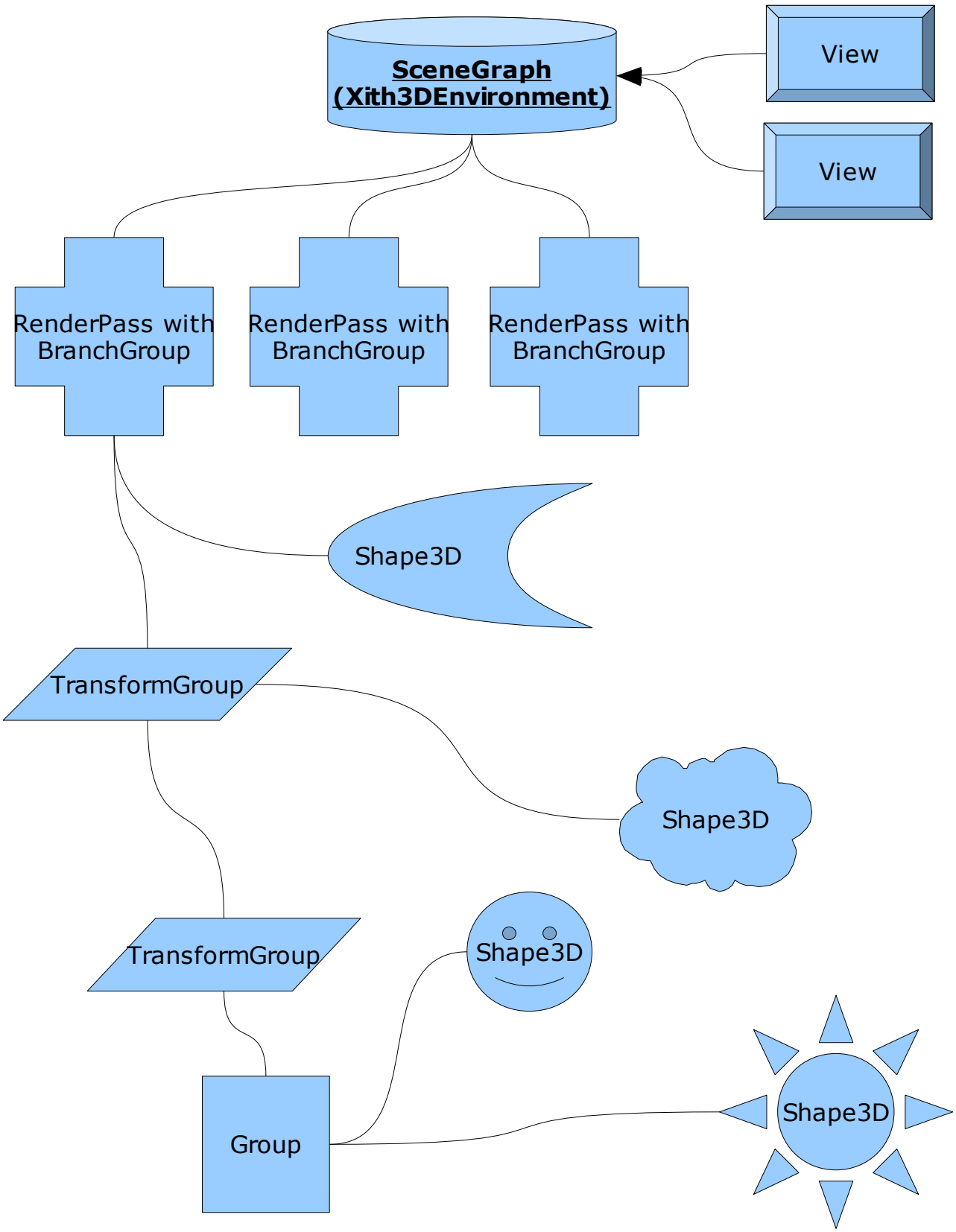
From root to leafs

The root of any SceneGraph in Xith3D is an instance of the class SceneGraph, which will in most cases be a Xith3DEnvironment. Apart from defining the root it is the interface to the used Renderer. It is responsible for invoking the Renderer to render the next frame.

And we need a way to define the point of view we want to render the scene from. This is done by the View class, which is simply instantiated and set up with its location, view vector and up vector and added to the SceneGraph by invoking the addView() method. (One instance is always automatically created with the SceneGraph.)

The actual tree starts with a root group, which is called BranchGroup in Xith3D. A BranchGroup is always bound to a certain RenderPass, which can have various configuration settings. One BranchGroup can be attached to any number of RenderPasses. Create and add a RenderPass together with a BranchGroup and default settings with e.g. the addPerspectiveBranch() method of the SceneGraph.

Now you can add any instance of the abstract Node class to the BranchGroup. Generally there are two main types of Nodes: Groups and Leafs. There's the Group class which defines a simple group to hold other Nodes. Apart from grouping Nodes a Group also optimizes the rendering performance in a way described later. And there's the TransformGroup class described above. The most used Leaf type is the Shape3D class, which is also shortly described above.


SceneGraph
(Xith3DEnvironment)
View
View
RenderPass with
BranchGroup
RenderPass with
BranchGroup
RenderPass with
BranchGroup
Shape3D
TransformGroup
Shape3D
TransformGroup
Shape3D
Group
Shape3D

Strict tree

In Xith3D the SceneGraph's tree is strict. This means, that no Node can be added to more than one parent Group. But sometimes you want to render the exact same Shape3D multiple times. So to not create the same Shape3D multiple times you can use the sharedCopy() method to create a cheap, shared copy of the Shape3D. All Shape3D instances created this way will share their Appearance and Geometry instances. In case of Models you should use the sharedInstance() method instead.

SceneGraph optimization

As you may see not all shapes are visible at any time. If you look into some direction, all shapes behind you are not visible.
The view defines a frustum, and all Nodes (Groups and Leafs) outside this frustum are invisible. Invisible shapes don't need to be sent to OpenGL and can be sorted out to optimize performance. Happily we don't need to decide for each pixel, if it gets rendered (like OpenGL needs to do), nor we need to decide it for each triangle, but we can decide it per Shape3D and even per Group. All Nodes (including Groups) have a bounding sphere around them. If the bounding sphere is totally outside the view frustum, none of the contained child nodes need to be checked for containment nor sent to OpenGL. Remember, that the SceneGraph is traversed recursively to determine the Shape3Ds to be rendered and the particular transformation matrices.

So if you group some “near” Shape3Ds in one Group Node and add these Group Nodes to a parent group, these decisions can be made more quickly and your scene will be rendered in higher FPS.

The Switch Node is another possibility to gain performance. It is a Group Node, which is controlled by a bitmask. Only child Nodes with a bit of 1 will be traversed and rendered.

Multipass Rendering

Sometimes you want to ensure, that one part of the scene is rendered after another one. Maybe you also want the second part to overpaint the first one. This can be realized through multipass rendering.

The BranchGroups are guaranteed to be rendered in the order, their owning RenderPasses are added to the SceneGraph.

If you want the second pass to overpaint the first one, you need to set the setLayeredMode() method of the Renderer to true.

Each RenderPass is set up by a RenderPassConfig, which can be accessed through the getConfig() method of the RenderPass.

Here is a simple example, how to build up a SceneGraph with Xith3DEnvironment:

```
01 public class SceneWithCube extends InputAdapterRenderLoop
02 {
03     @Override
04     public void onKeyReleased( KeyReleasedEvent e, Key key )
05     {
06         switch ( key.getKeyID() )
07         {
08             case ESCAPE:
09                 this.end();
10                 break;
11         }
12     }
13 
14     private BranchGroup createScene() throws Exception
15     {
16         ResourceLocator resLoc = ResourceLocator.create( "test-resources/" );
17         resLoc.createAndAddTSL( "textures" );
18 
19         // org.xith3d.geometry.Cube is a Shape3D extension.
20         Cube cube = new Cube( 3.0f, "stone.jpg" );
21 
22         TransformGroup tg = new TransformGroup();
23         tg.getTransform().rotX( FastMath.PI );
24         tg.addChild( cube );
25 
26         BranchGroup bg = new BranchGroup();
27         bg.addChild( tg );
28 
29         return( bg );
30     }
31 
32     public SceneWithCube() throws Exception
33     {
34         super( 128f ); // Limit the FPS to 128
35 
36         // create a new SceneGraph implementation
37         Xith3DEnvironment env = new Xith3DEnvironment( this );
38 
39         // create a Canvas3D and add it to the environment
40         Canvas3D canvas = Canvas3DFactory.createWindowed( 800, 600,
41                                                 "Scene with cube" );
42 
43         env.addCanvas( canvas );
44 
45         // register input devices
46         InputSystem.getInstance().registerNewKeyboardAndMouse( canvas.getPeer() );
47 
48         // add the BranchGroup to the SceneGraph together with a new
49         // RenderPass, which is rendered in perspective projection
50         env.addPerspectiveBranch( createScene() );
51 
52         // start the RenderLoop
53         this.begin();
54     }
55 
56     public static void main( String[] args ) throws Exception
57     {
58         new SceneWithCube();
59     }
60 }
```