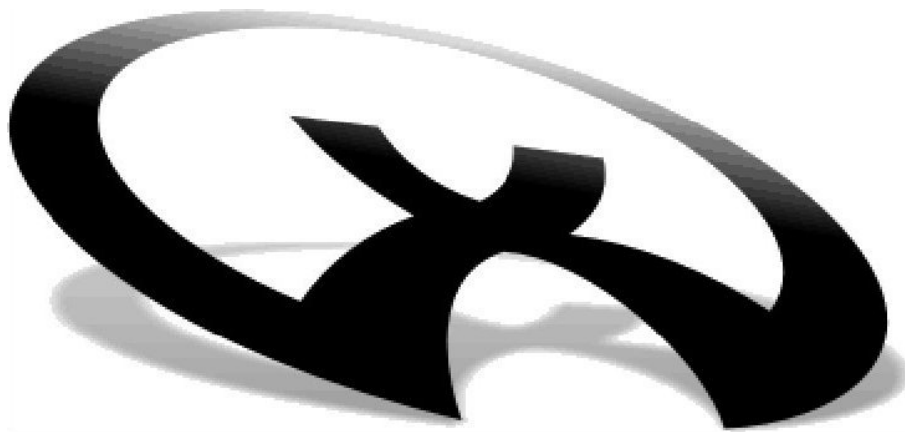


Xith3D in a Nutshell

1st edition



Authors: *Amos Wenger, Marvin Fröhlich*

October 4, 2006

Contents

	Prologue: Xith3D history	3
	Prologue: Featured games	4
0	Quick Rush	7
1a	Our first empty scene	8
1b	Ease this code	9
2	Adjusting the camera	10
3	Catching input events	11
4	Easy FPS counter	12
5	The first shape	13
6	Lights	15
7a	Animation (Rotation)	19
7b	Easy Rotation	20
7c	Easy Rotation – an alternative way	20
8a	Thread safe operations	22
8b	Intervals	23
9a	Picking	24
9b	Alternative Picking	25
10	Screenshots	26
11	3D Models	27
12	Choosing an OpenGL layer	29
13	Multipass rendering	30
14a	HUD	31
14b	More Widgets	34
14c	Theming the HUD	38
15	Swing integration	39
	Epilogue	40

Prologue: Xith3D history

Xith3D has been created in 2003 by David Yazel for his game, Magicosm. It's born from the frustration developers had with Java3D (which was developed by Sun, and has now been public-sourced), which Xith3D is heavily inspired of.



A screenshot of Magicosm, the game David Yazel developed Xith3D for.

Soon others joined David in development and the project has grown, presenting to you now a full-featured 3D scenegraph designed specially for games.

Development of Magicosm was discontinued and David stopped programming. William Denniss is now the official maintainer of Xith3D. He used to manage the community site, which has been replaced by a new site in 2006, thanks to Amos Wenger. Known past or present Xith3D developers are: David Yazel, William Denniss, Yuri Vl. Guschin, Kevin Glass, Hawkwind, Jens Lehman, David Wallace Croft, Arne Müller, Lilian Chamontin, Amos Wenger and Marvin Fröhlich... (if we forgot anyone, let us know on Xith3D forum).

Xith3D development is now managed primarily on our site : <http://www.xith.org/> especially on the forum. You can even publish your game on xith.org, if you want to (just ask on the forum).

Prologue: Featured Games

There're some cool games made with Xith3D. Some are unfinished, others more tech demos than anything else. But they're good showcases of what is possible with Xith3D.

Martian Madness

Made by Kevin Glass as a mean to improve his Xith3D knowledge. It's pretty fun to play, although incomplete.



<http://www.cokeandcode.com/node/307>

Jack Flowers

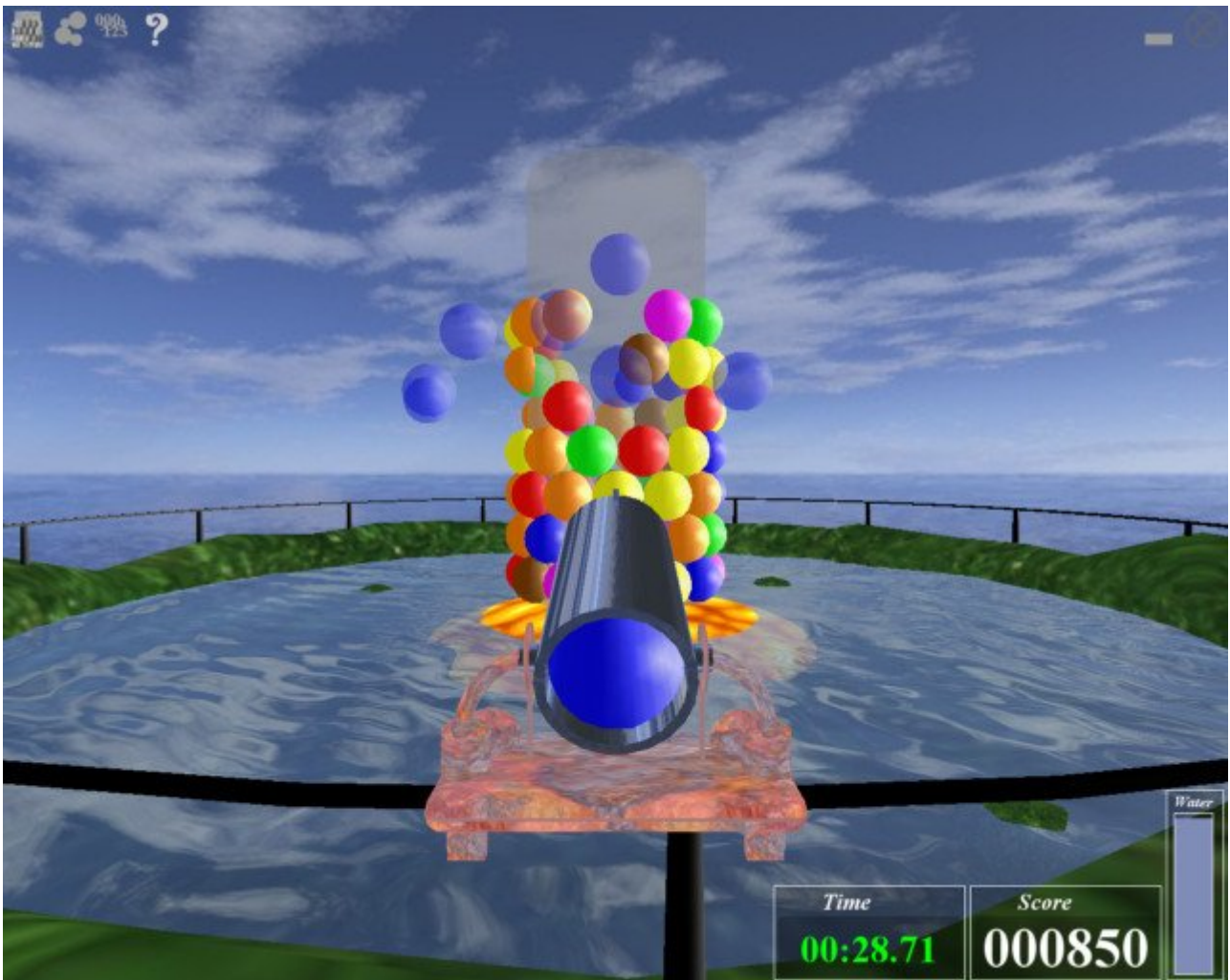
Recently beta-released, this game has been coded by Lilian Chamontin, a French hobbyist developer. Even though it's a small game, hours of great fun are guaranteed.



<http://www.javapause.com/games/jack/>

Zplax!

Nice shooting puzzle game realized by Alistair Dickie. Released as a shareware, costs only 10\$:-)



<http://www.alistairdickie.com/>

Chapter 0 - Quick Rush

Now let's go for the coding side.

All following examples assume to have a main method to start our code like this:

```
01 public class EmptyScene
02 {
03     public EmptyScene()
04     {
05         ...
06     }
07
08     public static void main(String[] args)
09     {
10         new EmptyScene();
11     }
12 }
```

Chapter 1a - Our first empty scene

Setting up an empty Scene is really easy, thanks to some wrapper classes in `org.xith3d.render` and subpackages. Be sure to have a recent Xith3D version greater than 0.8.0.

Here is a simple example of how to set up your first (empty) scene:

```
01 public class EmptyScene
02 {
03     public EmptyScene()
04     {
05         Xith3DEnvironment env = new Xith3DEnvironment();
06
07         Canvas3DWrapper canvas = Canvas3DWrapper.createStandalone(
08                                     Resolution.RES_800X600,
09                                     "My empty scene" );
10
11         env.addCanvas( canvas );
12
13         RenderLoop rl = new RenderLoop();
14         rl.setMaxFPS( 128L );
15         rl.addRenderEngine( env );
16
17         // never forget to start the RenderLoop!
18         rl.begin();
19     }
20 }
```

So what does all this mean?

First of all you need the Xith3D basics which are created and handled by `Xith3DEnvironment` (line 05). Then you need an area to draw the scene on. This is created in line 07. The window will have a resolution of 800x600 and the title will be "My empty scene". Check the other factory methods of `Canvas3DWrapper` and the Constructor, too.

This `Canvas3D` needs to be added to the environment (line 10), so that Xith3D knows where to draw the rendered scene on.

The scene is rendered frame by frame in a more or less constant loop. To let our further code work besides this rendering loop it has to run in a separate thread. And since our Operating System probably is multi threaded we'll have to give the other threads a chance to work. Therefore we want to limit the maximum FPS to 128, which will also avoid a CPU load of 100%.

But don't worry. You don't have to do it on your own. Just create an instance of `RenderLoop` and call the `setMaxFPS(long)` method like in line 12 and 13.

Now this loop needs to be linked with the a `RenderEngine`, which is our `Xith3DEnvironment`. This is done in line 13. Then just let the `RenderLoop` work and start it like in line 16. Note, that you can pass a boolean to the `begin()` method of `RenderLoop` telling if the loop shall run in a separate thread.

The result of this coding will be an 800x600 sized window with the dark grayed `Canvas3D` on it rendered at 128 FPS (max).

Chapter 1b - Ease this code

This example is also possible in a little less lines of code like this:

```
01 public class EmptyScene
02 {
03     public EmptyScene()
04     {
05         RenderLoop rl = new RenderLoop( 128L );
06
07         Xith3DEnvironment env = new Xith3DEnvironment( rl );
08
09         env.addCanvas( Canvas3DWrapper.createStandalone( Resolution.RES_800X600,
10                                                         "My empty scene"
11                                                         ) );
12         rl.begin();
13     }
14 }
```

To make easy use of some advanced features of the RenderLoop we will let our EmptyScene class extend the RenderLoop class which should always be the preferred way. These advanced features are available through listeners, too, but extending and overriding is just easier.

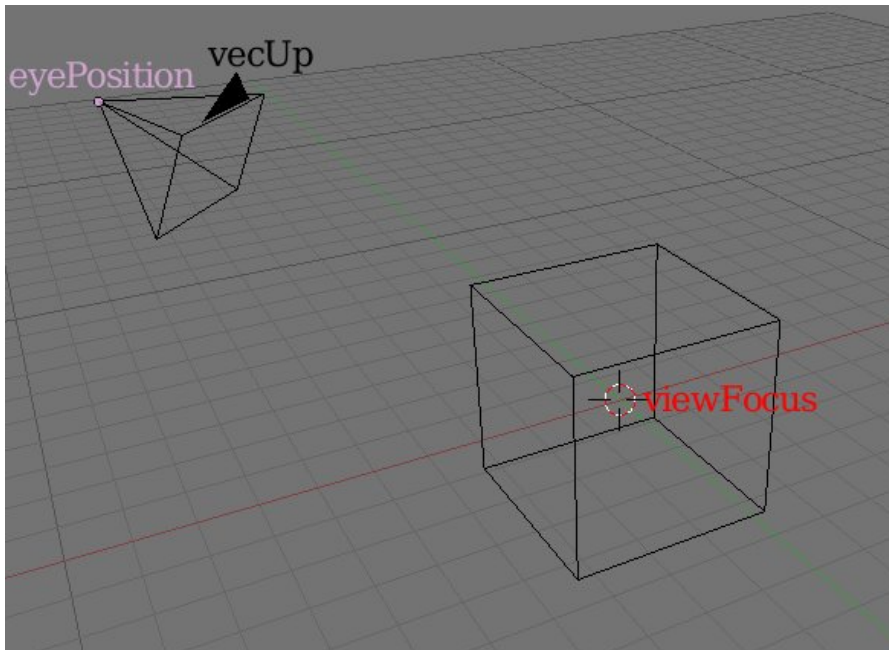
```
01 public class EmptyScene extends RenderLoop
02 {
03     public EmptyScene()
04     {
05         super( 128L );
06
07         Xith3DEnvironment env = new Xith3DEnvironment( this );
08
09         env.addCanvas( Canvas3DWrapper.createStandalone( Resolution.RES_800X600,
10                                                         "My empty scene"
11                                                         ) );
12
13         this.begin();
14     }
15 }
```

Chapter 2 - Adjusting the camera

The camera needs a position in space where it's placed and one to focus at and a vector pointing up to define the camera's rotation. This is simply done by adding three more parameters to the Xith3DEnvironment constructor or invoking the env.getView().lookAt(Tuple3f, Tuple3f, Tuple3f) method.

The Parameters are all the same three ones:

- eyePosition: The position of the camera (or where you look from)
- viewFocus: The center focus point (or where you look at)
- vecUp: The vector pointing up (or how far you lean to a side)



```
01 public class EmptyScene extends RenderLoop
02 {
03     public EmptyScene()
04     {
05         super( 128L );
06
07         Tuple3f eyePosition = new Vector3f( 0.0f, 0.0f, 5.0f );
08         Tuple3f viewFocus = new Vector3f( 0.0f, 0.0f, 0.0f );
09         Tuple3f vecUp = new Vector3f( 0.0f, 1.0f, 0.0f );
10
11         Xith3DEnvironment env =
12             new Xith3DEnvironment( eyePosition, viewFocus, vecUp, this );
13
14         // or alternatively...
15         env.getView().lookAt( eyePosition, viewFocus, vecUp );
16
17         env.addCanvas( Canvas3DWrapper.createStandalone( Resolution.RES_800X600,
18                                                         "My empty scene"
19                                                         ) );
20         this.begin();
21     }
22 }
```

Note: The parameters chosen for eyePosition, viewFocus and vecUp are the ones used as default by Xith3DEnvironment and we won't use them explicitly in the following chapters.

Chapter 3 - Catching input events

Having our class like this we can start overriding methods from RenderLoop. So we want to enable the user to quit the application by pressing the ESCAPE key or the left mouse button. Too easy...

```
01 public class EmptyScene extends RenderLoop
02 {
03     protected void onKeyReleased(int key)
04     {
05         switch (key)
06         {
07             case KeyCode.VK_ESCAPE:
08                 this.end();
09                 break;
10         }
11     }
12
13     protected void onMouseButtonReleased(int button, int x, int y)
14     {
15         switch (button)
16         {
17             case MouseCode.LEFT_BUTTON:
18                 this.end();
19                 break;
20         }
21     }
22
23     public EmptyScene()
24     {
25         super( 128L );
26
27         Xith3DEnvironment env = new Xith3DEnvironment( this );
28
29         Canvas3DWrapper canvas = Canvas3DWrapper.createStandalone(
30                                     Resolution.RES_800X600,
31                                     "My empty scene" );
32         this.registerKeyboardAndMouse( canvas );
33
34         env.addCanvas( canvas );
35
36         this.begin();
37     }
38 }
```

The two new methods reside in the RenderLoop class and are overridden in our EmptyScene class. They are called when an input event occurred.

Of course we need to attach input devices to the system, which is done in line 32.

Another way would be to add KeyboardListener, MouseListener or InputListener to your MouseDevice and KeyboardDevice instances, which can also be retrieved from the RenderLoop instance. Please refer to the HIAL manual for further documentation.

Chapter 4 - Easy FPS counter

Let's see how many FPS (frames per second) your machine can achieve with an empty scene. For that purpose you should remove the FPS limit.

```
01 public class EmptyScene extends RenderLoop
02 {
03     private Canvas3DWrapper canvas;
04
05     protected void onKeyReleased(int key)
06     {
07         switch (key)
08         {
09             case KeyCode.VK_ESCAPE:
10                 this.end();
11                 break;
12         }
13     }
14
15     protected void onMouseButtonReleased(int button, int x, int y)
16     {
17         switch (button)
18         {
19             case MouseCode.LEFT_BUTTON:
20                 this.end();
21                 break;
22         }
23     }
24
25
26     protected void onFPSCountIntervalHit(double fps)
27     {
28         canvas.setTitle( "My empty scene, FPS: " + (int)fps );
29     }
30
31     public EmptyScene()
32     {
33         super(); // Note: NO FPS limitation!
34
35         Xith3DEnvironment env = new Xith3DEnvironment( this);
36
37         Canvas3DWrapper canvas = Canvas3DWrapper.createStandalone(
38             Resolution.RES_800X600,
39             "My empty scene" );
40         this.canvas = canvas; // store this reference in a field
41         this.registerKeyboardAndMouse( canvas );
42
43         env.addCanvas( canvas );
44
45         this.begin();
46     }
47 }
```

The method `onFPSCountIntervalHit(double)` also resides in the `RenderLoop` class and is overridden as well. By default every half second this method is called, so printing out the current FPS won't be a performance issue. You can change this interval by calling the `setFPSCountInterval(long)` method on the `RenderLoop` (`EmptyScene`).

Chapter 5 - The first shape

Now let's see, if we can put some nice Shapes in our scene...

```
01 public class SceneWithCube extends RenderLoop
02 {
03     protected void onKeyReleased(int key)
04     {
05         switch (key)
06         {
07             case KeyCode.VK_ESCAPE:
08                 this.end();
09                 break;
10         }
11     }
12
13     private BranchGroup createScene()
14     {
15         Cube cube = new Cube( 3.0f, false );
16
17         Texture tex = TextureLoader.getInstance().getTexture( "stone.jpg" );
18
19         Appearance app = new Appearance();
20         app.setTexture( tex );
21         cube.setAppearance( app );
22
23         BranchGroup bg = new BranchGroup();
24         bg.addChild( cube );
25
26         return( bg );
27     }
28
29     public SceneWithCube()
30     {
31         super( 128L );
32
33         Xith3DEnvironment env = new Xith3DEnvironment( this );
34
35         Canvas3DWrapper canvas = Canvas3DWrapper.createStandalone(
36                                     Resolution.RES_800X600,
37                                     "My empty scene" );
38         this.registerKeyboardAndMouse( canvas );
39
40         env.addCanvas( canvas );
41
42         TextureLoader.getInstance().addTextureStreamLocator(
43             new TextureStreamLocatorFile( "demo/textures" ) );
44
45         env.addPerspectiveBranch( createScene() );
46
47         this.begin();
48     }
49 }
```

In line 42 we tell the singleton instance of TextureLoader to take “./demo” as a search path for textures. You MUST always do this before you load any texture.

The new createScene() method creates a Cube and applies a texture to it.

Textures are not applied directly to a Shape3D but to an instance of Appearance which is then applied to the Shape3D. The Appearance class has several other attributes which all together describe how the shape looks like.

Most of the classes in org.xith3d.geometry directly take the Texture reference, which enables us to ease the createScene() method a little:

```
01 public class SceneWithCube extends RenderLoop
02 {
03     protected void onKeyReleased(int key)
04     {
05         switch (key)
06         {
07             case KeyCode.VK_ESCAPE:
08                 this.end();
09                 break;
10         }
11     }
12
13     private BranchGroup createScene()
14     {
15         Cube cube = new Cube( "stone.jpg", false, 3.0f );
16
17         BranchGroup bg = new BranchGroup();
18         bg.addChild( cube );
19
20         return( bg );
21     }
22
23     public SceneWithCube()
24     {
25         super( 128L );
26
27         Xith3DEnvironment env = new Xith3DEnvironment( this );
28
29         Canvas3DWrapper canvas = Canvas3DWrapper.createStandalone(
30                                     Resolution.RES_800X600,
31                                     "My empty scene" );
32         this.registerKeyboardAndMouse( canvas );
33
34         env.addCanvas( canvas );
35
36         TextureLoader.getInstance().addTextureStreamLocator(
37             new TextureStreamLocatorFile( "demo/textures" ) );
38
39         env.addPerspectiveBranch( createScene() );
40
41         this.begin();
42     }
43 }
```

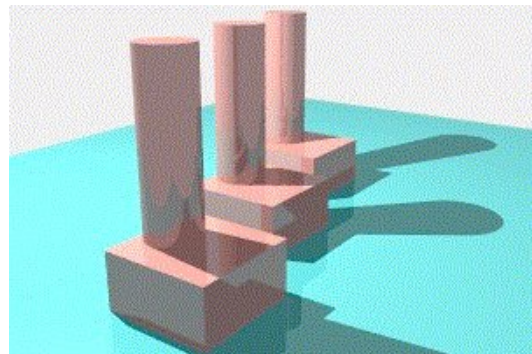
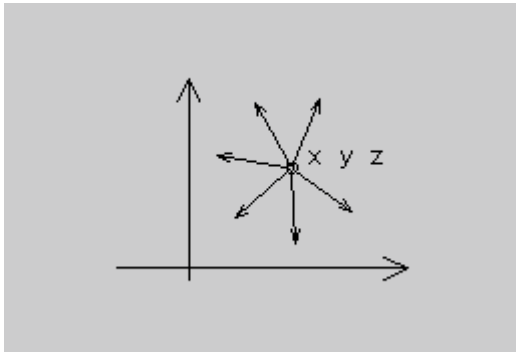
Chapter 6 - Lights

Let there be light! Did you notice, that all shapes had the same, homogeneous light level? This is ok for e.g. a geometry show case program, but for your action-drama game you probably want more. You can use up to eight light sources in Xith3D (due to OpenGL limitations).

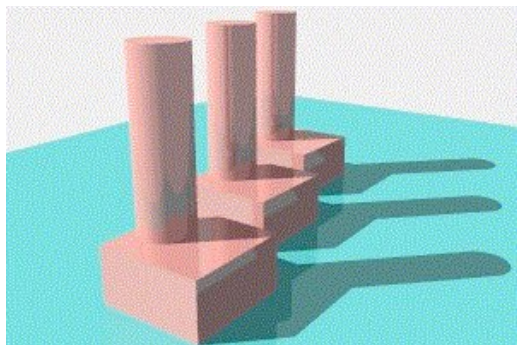
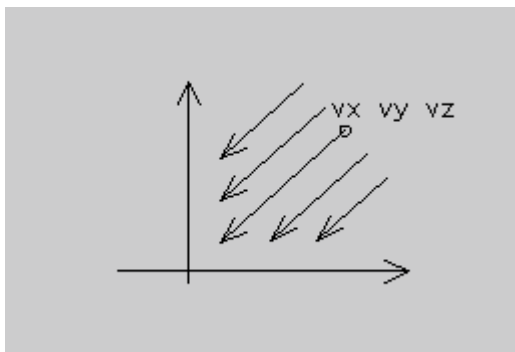
Light sources are set up by adding an instance of Light (actually a subtype) somewhere into the scenegraph. Any Shape3D node located in the same group node, where the Light node resides (or is a grandchild of this group) will be affected by this light source.

Types of lights are:

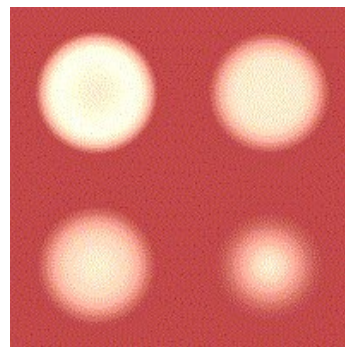
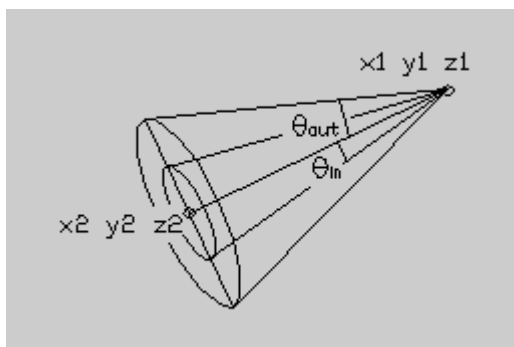
- AmbientLight: It illuminates your whole scene in a uniform way. You can adjust the color of the light. It's a good practice to always have a gray (0.3f, 0.3f, 0.3f) light in your game to have everything basely illuminated and you'll see (by mistake) not illuminated shapes.
- PointLight: Light emitting point. You can adjust the position (Point3f), attenuation (Tuple3f, values like 0.001f, 0.001f, 0.001f are a good value to start with) and of course color. (Playing with point lights is pure fun.)



- DirectionalLight: It defines an oriented light source with an origin at infinity. You can adjust the orientation (default is 0f, 0f, -1f) and color.



- SpotLight: Ideal for your party game :-). It defines a point light source located at some position in space and radiating in a specific direction. You can adjust direction (default see DirectionalLight), spread angle and concentration.



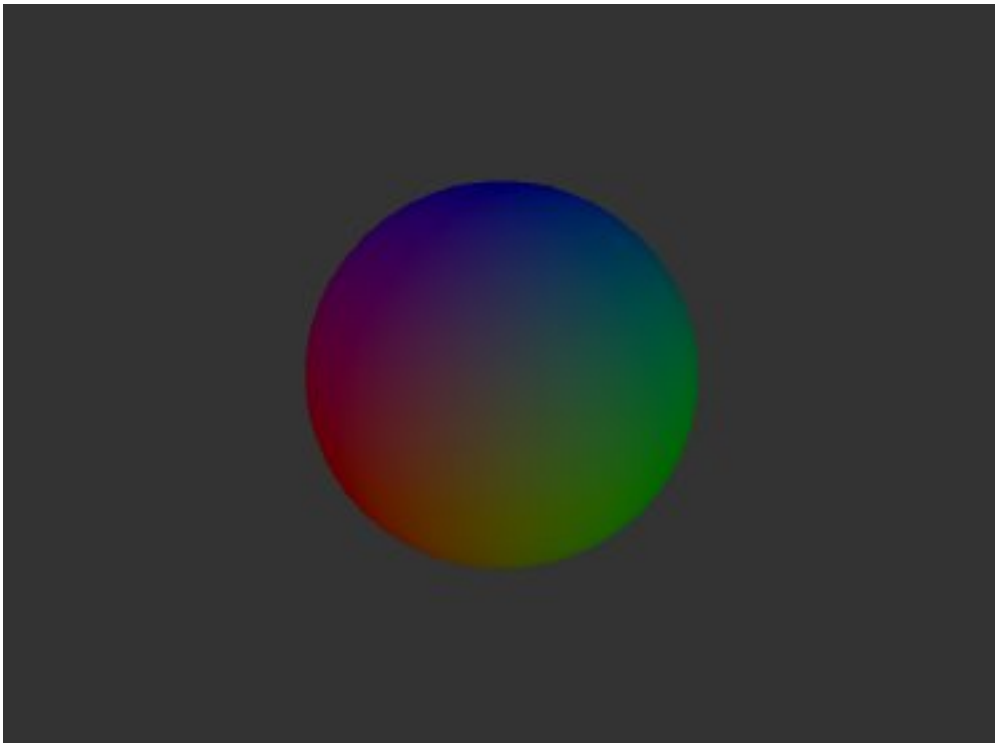
Lights - an example

The example is a pullout from a more complex one with three light sources. We will show how a GeoSphere is illuminated by one light source which can be dynamically switched on or off.

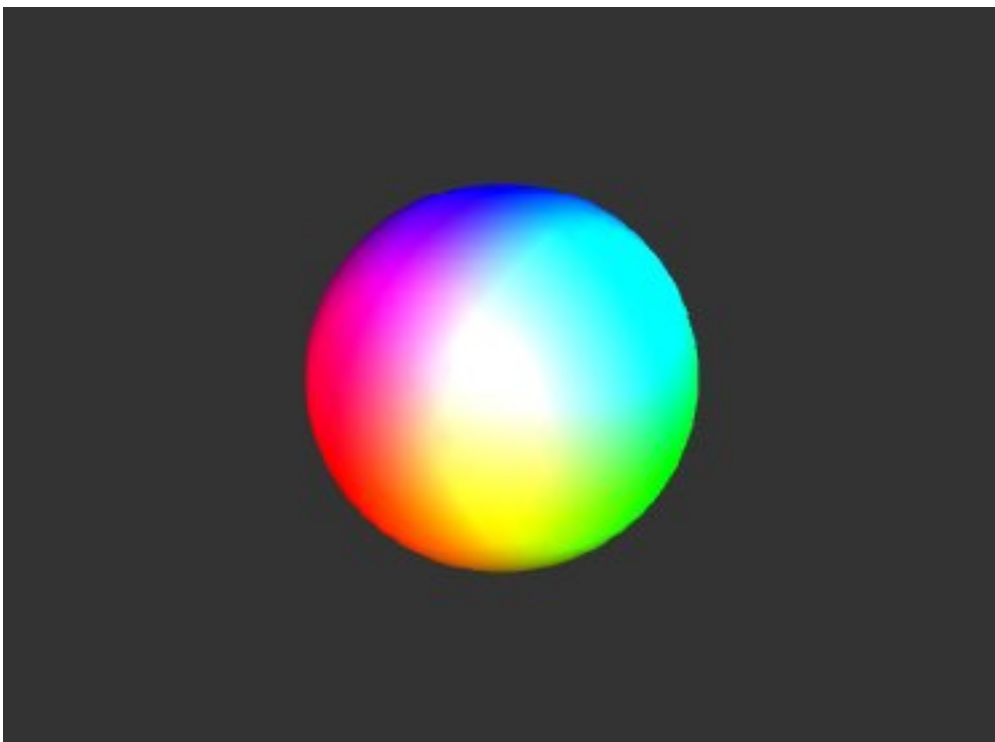
```
01 public class LightSources
02 {
03     ...
04
05     private BranchGroup createScene()
06     {
07         BranchGroup bg = new BranchGroup();
08
09         // Add a GeoSphere
10         GeoSphere sph = new GeoSphere( 8, GeometryArray.COLOR_3 |
11                                     GeometryArray.NORMALS, 3f );
12         bg.addChild( sph );
13
14         // Adjust appearance
15         Appearance app = new Appearance();
16         app.setMaterial( new Material( Color3f.WHITE, Color3f.BLACK,
17                                     Color3f.WHITE, Color3f.BLACK,
18                                     0.8f, true, Material.AMBIENT, true ) );
19         sph.setAppearance( app );
20
21         // Add some lighting
22         Light light = new PointLight( new Color3f( 0f, 1f, 0f ),
23                                     new Point3f( 10f, -5f, 5f ),
24                                     new Point3f( 0.005f, 0.005f, 0.005f )
25                                     );
26         bg.addChild( light );
27
28         // To switch off...
29         light.setEnabled( false );
30         // And to switch on...
31         light.setEnabled( true );
32
33         return( bg );
34     }
35
36     ...
37 }
```

But what are these strange colors of the Material? Here is a short summary:

- Diffuse color: The RGB color of the material when illuminated. The range of values is [0f, 1f]. The default diffuse color is (1f, 1f, 1f).
- Specular color: The RGB specular color of the material (highlights). The range of values is [0f, 1f]. The default specular color is (1f, 1f, 1f).
- Emissive color: The RGB color of the light the material emits, if any. The range of values is [0f to 1f]. The default emissive color is (0f, 0f, 0f).
- Shininess: The material's shininess, in the range [1f, 128f] with 1f being not shiny and 128f being very shiny. Values outside this range are clamped. The default value for the material's shininess is 64f.



Without light



With light

Chapter 7a - Animation (Rotation)

Nice, isn't it? But wouldn't it be even nicer, if the cube was rotating?

```
01 public class SceneWithRotatingCube extends RenderLoop
02 {
03     private Transform3D t3d;
04     private TransformGroup tg;
05     private AngleInterpolator angleX;
06     private AngleInterpolator angleY;
07
08     private BranchGroup createScene()
09     {
10         Cube cube = new Cube( "stone.jpg", false, 3.0f );
11
12         t3d = new Transform3D();
13         tg = new TransformGroup( t3d );
14
15         tg.addChild( cube );
16
17         return( new BranchGroup( tg ) );
18     }
19
20     protected void onRenderLoopStarted()
21     {
22         angleX = new AngleInterpolator( 0f, 1f, 0f, (float)Math.PI * 2f, true );
23         angleY = new AngleInterpolator( 0f, 1f, 0f, (float)Math.PI * 2f, true );
24
25         angleX.startIncreasing( this.getGameTime() );
26         angleY.startIncreasing( this.getGameTime() );
27     }
28
29     protected void loopIteration(long gameTime, long frameTime)
30     {
31         t3d.rotXYZ( angleX.getValue( gameTime ), angleY.getValue( gameTime ), 0f );
32         tg.setTransform( t3d );
33
34         super.loopIteration( gameTime, frameTime );
35     }
36
37     public SceneWithRotatingCube()
38     {
39         super( 128L );
40
41         Xith3DEnvironment env = new Xith3DEnvironment( this );
42
43         Canvas3DWrapper canvas = Canvas3DWrapper.createStandalone(
44                                     Resolution.RES_800X600,
45                                     "My empty scene" );
46         this.registerKeyboardAndMouse( canvas );
47
48         ...
49     }
50 }
```

The methods `onRenderLoopStarted()` and `loopIteration()` are again overridden in the `RenderLoop` class. `onRenderLoopStarted()` is called before the `RenderLoop` actually starts working and `loopIteration()` is called each loop iteration (each frame). Now our cube is not directly added to the environment (scenegraph), but added to a `TransformGroup`, which handles the rotation in this case and this `TransformGroup` is then added to the scenegraph. The two `AngleInterpolaters` interpolate the rotation angle at the current game time.

Chapter 7b - Easy Rotation

Nice but too complicated? Well we can do easier. But first we need to switch from `RenderLoop` to **ExtRenderLoop**:

```
01 public class SceneWithRotatingCube extends ExtRenderLoop
02 {
03     private BranchGroup createScene(Animator animator)
04     {
05         Cube cube = new Cube( "stone.jpg", false, 3.0f );
06
07         TransformationDirectives rotDirecs =
08             new TransformationDirectives( 0.3f, 0.2f, 0f );
09         RotatableGroup rg = new RotatableGroup( rotDirecs );
10
11         rg.addChild( cube );
12         animator.addAnimatableObject( rg );
13
14         return( new BranchGroup( rg ) );
15     }
16
17     public SceneWithRotatingCube()
18     {
19         super( 128L );
20
21         Xith3DEnvironment env = new Xith3DEnvironment( this );
22
23         Canvas3DWrapper canvas = Canvas3DWrapper.createStandalone(
24             Resolution.RES_800X600,
25             "My empty scene" );
26         this.registerKeyboardAndMouse( canvas );
27         env.addPerspectiveBranch( createScene( this ) );
28         ...
29     }
30 }
```

Now this is easy, isn't it? Specially notice line 12, where we register our `RotatableGroup`, which implements `Animatable`, as an animatable object to our `Animator` and immediately start the rotation. `ExtRenderLoop` is an `Animator`.

Check, if you can make further use of the `Animatable` interface.

Always refer to the `JavaDoc` for specific parameters.

Chapter 7c - Easy Rotation – an alternative way

In the last Example RotatableGroup is used. There's also TranslatableGroup and both classes extend AnimatableGroup.

It is also possible to directly use the GroupRotator or GroupTranslator, which are used by AnimatableGroup-extensions and which extend GroupAnimator:

```
01 public class SceneWithRotatingCube extends ExtRenderLoop
02 {
03     private BranchGroup createScene(Animator animator)
04     {
05         Cube cube = new Cube( "stone.jpg", false, 3.0f );
06
07         TransformGroup tg = new TransformGroup();
08         tg.addChild( cube );
09
10         TransformationDirectives rotDirecs =
11             new TransformationDirectives( 0.3f, 0.2f, 0f );
12         GroupRotator gr = new GroupRotator( tg, rotDirecs );
13
14         animator.addAnimatableObject( gr );
15
16         return( new BranchGroup( tg ) );
17     }
18
19     ...
20 }
```

Here a regular TransformGroup is used and passed to an instance of GroupRotator, which handles the rotation and which is itself passed to animator.addAnimatableObject(Animatable). In some cases this could be more usable. Just decide case by case, what you prefer.

Chapter 8a - Thread safe operations

If you want an operation to be done thread safely by the render loop, you can make use of the ScheduledOperation interface.

Implement it like this:

```
01 private class MyOperation implements ScheduledOperation
02 {
03     private boolean isAlive = true;
04
05     public boolean isPersistent()
06     {
07         return( true );
08     }
09
10     public void setAlive(boolean alive)
11     {
12         this.isAlive = alive;
13     }
14
15     public boolean isAlive()
16     {
17         return( isAlive );
18     }
19
20     public void executeOperation(long gameTime, long frameTime)
21     {
22         // your code for the operation
23     }
24 }
```

The executeOperation method will be called by the render thread each loop iteration before the rendering is performed until either the isAlive() method or isPersistent() returns *false*. The isAlive() method is not checked for non persistent operations. When an operation is not persistent or alive it will be removed from the scheduler the next loop iteration.

There's also an abstract ScheduledOperationImpl class, which you can use instead to save some coding.

Now when you have a working environment with an ExtRenderLoop you can easily add this operation to the scheduler (ExtRenderLoop implements OperationScheduler) and it will be magically done each loop iteration.

```
01 public class MySceneWithSchedOp extends ExtRenderLoop
02 {
03     public MySceneWithSchedOp()
04     {
05         super( 128L );
06
07         ...
08
09         this.scheduleOperation( new MyOperation2() );
10
11         ...
12     }
13 }
14 }
```

Chapter 8b - Intervals

Maybe you want something to be done periodically. Easy with ExtRenderLoop:

```
01 public class MyScene extends ExtRenderLoop
02 {
03     private Appearance app;
04     private Texture[] textures;
05     private int currTex = 0;
06
07     private BranchGroup createScene()
08     {
09         Cube cube = new Cube( 3.0f, false );
10
11         textures = new Texture[] {
12             TextureLoader.getInstance().getTexture( "stone.jpg" ),
13             TextureLoader.getInstance().getTexture( "William.jpg" )
14         };
15
16         app = new Appearance();
17         app.setTexture( textures[ currTex ] );
18         cube.setAppearance( app );
19
20         ...
21     }
22
23     protected void onIntervalHit(Interval interval, long gameTime, long frameTime)
24     {
25         if (interval.getName().equals( "my interval" ))
26         {
27             app.setTexture( textures[ ++currTex % 2 ] );
28             // call interval.kill() to stop and remove the Interval
29         }
30     }
31
32     public MyScene()
33     {
34         super( 128L );
35
36         Xith3DEnvironment env = new Xith3DEnvironment( this );
37
38         ...
39
40         env.addPerspectiveBranch( createScene() );
41
42         this.addInterval( new Interval( 3000L, "my interval" ) );
43
44         this.begin();
45     }
46 }
```

We create an instance of Interval (line 42) which is hit every three seconds (3000 milliseconds) and has "my interval" as its name.

The method onIntervalHit() is overridden from the ExtRenderLoop class and is called, each time the interval is hit. The hit Interval instance as well as the current gameTime and frameTime are passed to the method. In line 25 we check, which Interval was hit and do the appropriate action. This action will of course be done from the render thread, and is therefore always thread safe.

Chapter 9a - Picking

So you want to pick the cube with your mouse? There are two ways to go:

```
01 public class SceneWithRotatingCube extends ExtRenderLoop
02 {
03     private PickEngine pickEngine;
04
05     protected void onMouseButtonPressed(int button, int x, int y)
06     {
07         PickResult pr = pickEngine.pickNearest( x, y );
08
09         if (pr != null)
10         {
11             System.out.println( "You picked a shape called " +
12                                 "\"" + pr.getShape().getName() + "\"." );
13         }
14         else
15         {
16             System.out.println( "You just picked nothing!" );
17         }
18     }
19
20     private BranchGroup createScene(Animator animator)
21     {
22         Cube cube = new Cube( "stone.jpg", false, 3.0f );
23         cube.setName( "my rotating cube" );
24
25         ...
26     }
27
28     public SceneWithRotatingCube()
29     {
30         super( 128L );
31
32         Xith3DEnvironment env = new Xith3DEnvironment( this );
33
34         this.pickEngine = env.getCanvas();
35
36         ...
37     }
38 }
```

Since picking MUST be done by the render thread in Xith3D, the picking operation needs to be synchronized with the rendering thread. In the above way this is automatically done by the engine.

Chapter 9b - Alternative Picking (preferred way)

The second way of picking is done with the help of PickingLibrary. In this way the synchronization needs to be made.

So you don't want to handle the synchronization on your own? Just switch from Xith3DEnvironment to **ExtXith3DEnvironment** and do the following:

```
01 public class MyScene extends ExtRenderLoop implements PickListener
02 {
03     private PickScheduler pickScheder;
04
05     public void onObjectsPicked(List<PickResult> pickResults,
06                               Object userObject, long pickTime) {}
07     public void onObjectPicked(PickResult nearest, Object userObject, long pickTime)
08     {
09         System.out.println( "You picked a shape called " +
10                             "\"" + nearest.getShape().getName() + "\"." );
11     }
12
13     public void onPickingMissed(Object userObject, long pickTime)
14     {
15         System.out.println( "You just picked nothing!" );
16     }
17
18     protected void onMouseButtonReleased(int button, int x, int y)
19     {
20         pickScheder.pick( x, y, this, false );
21     }
22
23     private BranchGroup createScene(Animator animator)
24     {
25         Cube cube = new Cube( "stone.jpg", false, 3.0f );
26         cube.setName( "my rotating cube" );
27
28         ...
29     }
30
31     public MyScene()
32     {
33         super( 128L );
34
35         ExtXith3DEnvironment env = new ExtXith3DEnvironment( this );
36         this.pickScheder = env;
37
38         ...
39     }
40 }
41
```

Isn't this easy? We do the picking on the ExtXith3DEnvironment, which handles the thread safety for you. When the picking is done (thread safely), one of the listener methods is called. Notice the last, boolean parameter in line 21, which tells the picking algorithm to pick only the nearest Shape3D in case of *false*. In this case the onObjectPicked method of the PickListener is called when the picking is successful rather than the onObjectsPicked method if this parameter was *true*. If the picking is not successful the onPickingMissed method is called when picking is complete.

Chapter 10 - Screenshots

In some cases you will want to take some screen shots of your rendered scene. This is fairly easy with or without ExtRenderLoop or ExtXith3DEnvironment.

```
01 public class MyScene extends ExtRenderLoop
02 {
03     private ScreenshotEngine shotEngn1;
04     private ScreenshotEngine shotEngn2;
05
06     protected void onKeyReleased(int key)
07     {
08         switch (key)
09         {
10             case KeyCode.VK_F1:
11                 shotEngn1.takeScreenshot( false );
12                 break;
13
14             case KeyCode.VK_F2:
15                 shotEngn2.takeScreenshot( false );
16                 break;
17         }
18     }
19
20     private BranchGroup createScene(Animator animator)
21     {
22         ...
23     }
24
25     public MyScene()
26     {
27         super( 128L );
28
29         Xith3DEnvironment env = new Xith3DEnvironment( this );
30
31         Canvas3DWrapper canvas = Canvas3DWrapper.createStandalone(
32                                     Resolution.RES_800X600,
33                                     "My empty scene" );
34         this.registerKeyboardAndMouse( canvas );
35
36         this.shotEngn1 = env;
37         this.shotEngn2 = canvas;
38
39         this.begin();
40     }
}
```

The two calls in the lines 11 and 15 are functionally equal and will create a new non-alpha-channel-screenshot in the current working directory. So you may choose one of them.

Please check the other takeScreenshot() method signatures and their JavaDoc.

Chapter 11 - 3D Models

Primitives aren't bad but we want to use these nifty 3D models you can find at <http://www.amazing3d.com/free/free.shtml>, for example.

Now let's make a quick point about 3D models file formats. You have several formats around there for 3D gaming:

<i>Name</i>	<i>Extension</i>	<i>Texture</i>	<i>Frame anim</i>	<i>Skeletal anim</i>	<i>Supported ?</i>
3DS Max BINARY	.3ds	X	X		Yes, but...
3DS Max ASCII	.ase	X	X		Yes
Wavefront OBJ	.obj	X			Yes
Quake 2	.md2	X	X		Yes
Quake 3	.md5	X	X	X	Has been done but not in the toolkit yet
Quake 3 Level	.bsp	X	N/A	N/A	yes
Cal3D	.cfg	X	X	X	Yes
Collada (1.4)	.dae	X	X	X	Yes

As you can see, Xith3D supports most of them. However we advise you to avoid 3DS Max BINARY. We often had issues with them and Wavefront OBJ is much, much better, although not supporting frame animation. So for animation, Quake 2 format is old, better choose: Cal3D is a must, MD5 fits nicely, too.

Now what're the tools you could advice your artists to work with?

- ~~3DS Max~~: We don't think it's a good choice: It's expensive (don't support piracy!) and import/export Plugins are hard to find, or commercial.
- Blender: The must-have. Can do pretty everything, though 80% of its features are useless for games, it will fit your needs with ease. You can find import/export scripts at <http://blender.org/>. Beware 3DS cannot currently be imported/exported with textures.
- Wings3D: Very good low-poly modeler. Doesn't support animations, but very powerful. Worth the try: <http://wings3d.org/>. You may want to create your model's shapes with Wings3D, then export them to OBJ and animate them with Blender.

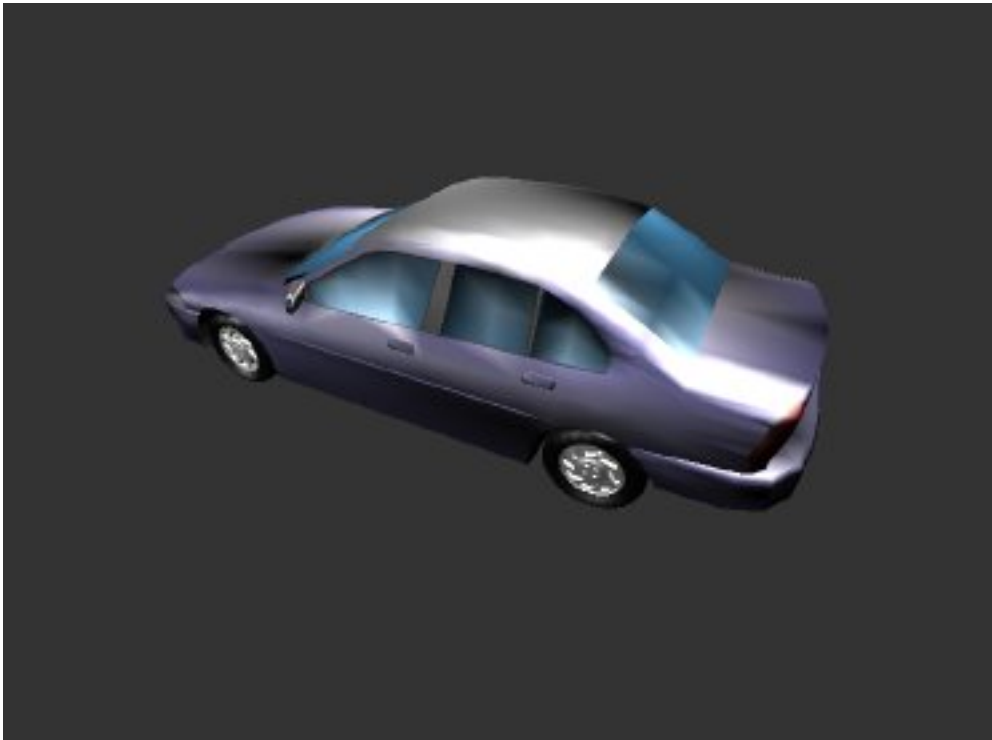


3D Models - an example

In our example, we use a car model named Mirage downloaded from a free web site and converted to Wavefront OBJ using Wings3D. As precedently, we just quote the interesting part of the source : model loading.

```
01 public class ModelLoadingExample
02 {
03     ...
04     private BranchGroup createScene() throws Exception
05     {
06         OBJModel model = new OBJLoader().loadModel( "Mirage.obj" );
07
08         return( new BranchGroup( model ) );
09     }
10
11     ...
12
13 }
```

The result will look as follows:



All model loaders extend an abstract common base class and are used the same way with slight differences in detail. The Model classes also extend a common base class and are therefore used the same way, too.

Chapter 12 - Choosing an OpenGL layer

Did you know, Xith3D is OpenGL layer independent? Currently JOGL (JSR-231) and LWJGL are supported. Even JOGL's bindings to AWT, Swing and SWT are all implemented and usable in Xith3D. LWJGL and JOGL/AWT both support fullscreen exclusive mode, though this support cannot be guaranteed in JOGL/AWT mode. On newer systems it will work (only with Java 6 on Linux).

- Why would you use JOGL (JSR-231)? It's supported originally by Sun and has great compatibility with existing hardware.
- Why would you use LWJGL? On some systems it runs faster (on others slower), but the difference is not very big. It has guaranteed fullscreen support.



But how do I use a specific one? By default, JOGL_AWT is used (maybe this default changes in a newer version), but it is really easy to switch. Just call a different factory method of Canvas3DWrapper. Remember our EmptyScene class. Modify it like this:

```
01 public class EmptyScene extends RenderLoop
02 {
03     public EmptyScene()
04     {
05         super( 128L );
06
07         Xith3DEnvironment env = new Xith3DEnvironment( this );
08
09         env.addCanvas( Canvas3DWrapper.createStandalone(
10                                     OpenGLLayer.LWJGL,
11                                     Resolution.RES_800X600,
12                                     "My empty scene"
13                                     ) );
14         this.begin();
15     }
16 }
```

Chapter 13 – Multipass rendering

Sometimes you will want to force the renderer to first render some parts of your scene and then (after it) render a second part. This is especially useful to render a HUD on top of you actual scene (described in chapter 14).

To achive this you need to create at least two BranchGroups and add them to the environment. These BranchGroups need to be attached to an instance of RenderPass with an appropriate RenderPassConfig.

Do it like this:

```
01 public class MultiPassRendering extends ExtRenderLoop
02 {
03     private BranchGroup createMainScene()
04     { ... }
05
06     private BranchGroup createParallelScene()
07     { ... }
08
09     public MultiPassRendering()
10     {
11         super( 128L );
12
13         Xith3DEnvironment env = new Xith3DEnvironment( this );
14
15         ...
16
17         RenderPassConfig scenePassConfig =
18             new RenderPassConfig( RenderPassConfig.PERSPECTIVE_PROJECTION );
19         env.addBranchGraph( createMainScene(), scenePassConfig );
20
21         RenderPassConfig paraPassConfig =
22             new RenderPassConfig( RenderPassConfig.PARALLEL_PROJECTION );
23         env.addBranchGraph( createParallelScene(), paraPassConfig );
24
25         ...
26     }
27 }
```

This way the main 3D scene is rendered first (first added to the environment) in (normal) perspective projection and the HUD branch is rendered second in parallel projection in front of the main scene. By default this is done in layered mode. This means, that all HUD stuff will overpaint the main scene. You can change this by invoking `env.getRenderer().setLayeredMode(boolean)`.

There're convenience methods to ease this coding:

```
01 env.addPerspectiveGraph( createMainScene() );
02 env.addParallelGraph( new BranchGroup( createParallelScene() ) );
```

or:

```
01 env.addPerspectiveGraph( createMainScene() );
02 env.addParallelGraph( createParallelScene() );
```

But keep in mind when using the `addParallelChild()` and `addPerspectiveChild()` methods, that they don't create separate RenderPasses for each call, but try to reuse a previously created one and only create a new one, when none was created before.

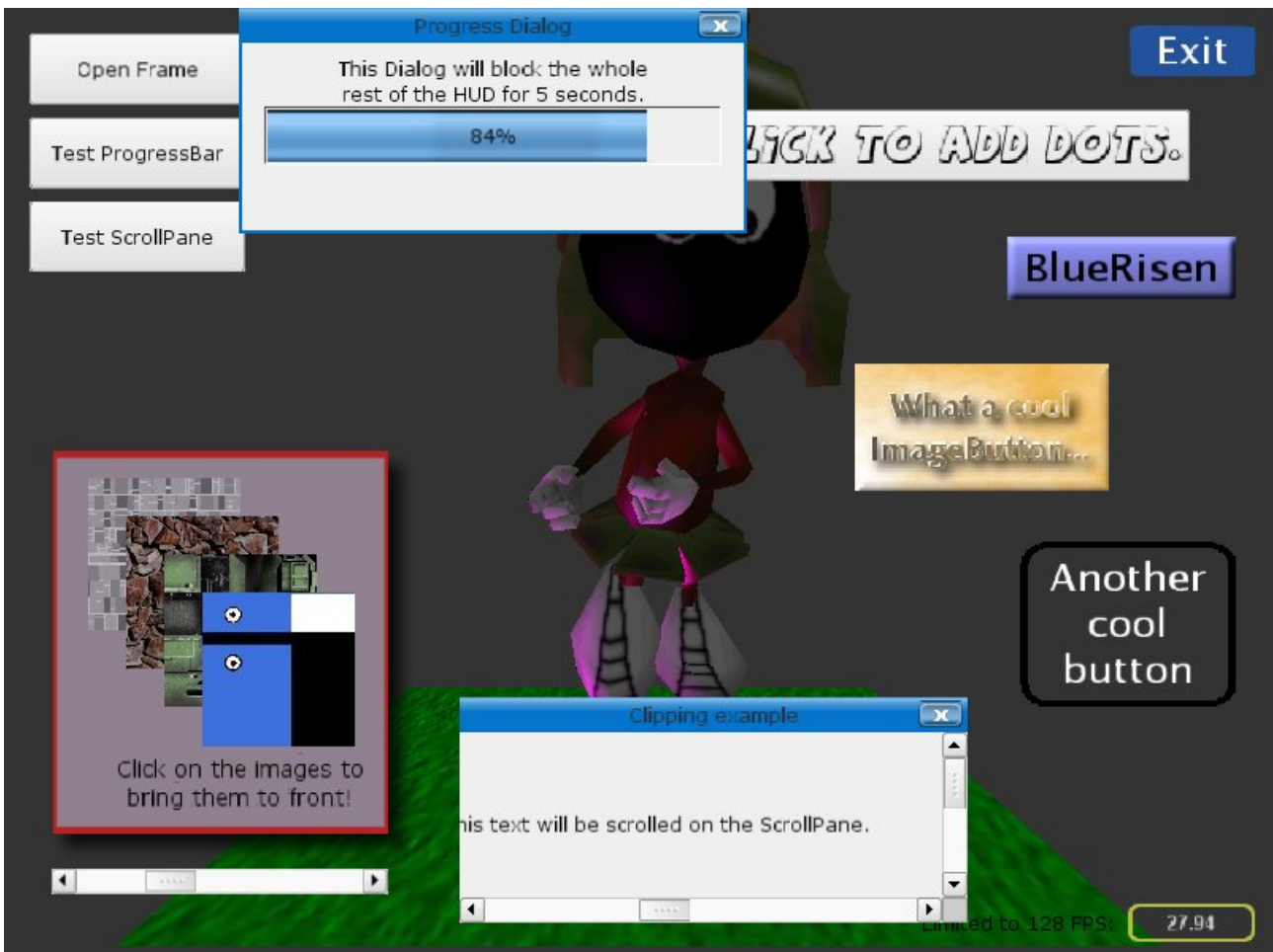
Have a look at the `RenderPassConfig` class, that is attached to each `RenderPass` instance. It offers several possibilities, that might be useful for you.

Chapter 14a - HUD

Maybe you want to have a HUD in your 3D world and can't or don't want to make use of Swing integration.

A HUD can even be rendered on top of your 3D scene (and it usually is).

In the `org.xith3d.ui.hud` package and subpackages there are several classes implementing a HUD functionality. Besides the `org.xith3d.ui.hud.HUD` class especially the classes in `org.xith3d.ui.hud.widgets` are important.



The HUD Test, showing clipped text, various themed buttons, an FPS counter, a scrollable panel, and a progress dialog.

An example:

```
01 public class MyHUD extends ExtRenderLoop implements ButtonListener
02 {
03     public void onPressed(Button button, Object userObject)
04     {
05         if (userObject.equals( "EXIT_BUTTON" ))
06         {
07             this.end();
08         }
09     }
10
11     private BranchGroup createMainScene()
12     {
13         ...
14     }
15
16     private HUD createHUD(Sized2i canvasSize)
17     {
18         HUD hud = new HUD( canvasSize, 800f, this );
19         // or: HUD hud = new HUD( canvasSize, this );
20
21         this.getInputManager().registerInputListener( hud );
22
23         Button button = new Button( 200f, 60f, "Click me to exit" );
24         button.setUserObject( "EXIT_BUTTON" );
25         button.addButtonListener( this );
26         hud.addWidgetCentered( button );
27
28         return( hud );
29     }
30
31     public MyHUD()
32     {
33         super( 128L );
34
35         Xith3DEnvironment env = new Xith3DEnvironment( this );
36
37         Canvas3D canvas = Canvas3DWrapper.createStandalone(
38                                 Resolution.RES_800X600,
39                                 "My HUD" );
40         env.addCanvas( canvas );
41
42         this.registerKeyboardAndMouse( canvas );
43
44         env.addPerspectiveGraph( createMainScene() );
45         // make sure, the HUD is added correctly like this:
46         env.addRenderPass( createHUD( canvas ).getRenderPass() );
47
48         this.begin();
49     }
50 }
```

You can give any Widget a z-index (through the constructor or the appropriate setter). This enables you to display one Widget in front of another.

So let's explain the HUD creation example...

First we create the HUD itself and choose a virtual resolution of 800x600 pixels. It is very important, that the virtual resolution has the same aspect ratio as the physical resolution. We only need to pass the horizontal (virtual) resolution to the HUD constructor and the vertical is automatically calculated. The last parameter of the HUD constructor is an instance of OperationScheduler, which is our ExtRenderLoop. The OperationScheduler's functionality is needed by some Widgets.

A HUD needs to retrieve input events which it will get from an InputListener implementation. The HUD implements this interface, so we simply need to link the HUD with our RenderLoop's InputManager (line 21). Without this line, you won't be able to effectively click on any Widget (like Button).

Now let's create a Widget. We take a Button as an example. Any Widget has a size and any Button can have a text and/or images to display the different parts of it in different states. The most simple case of an image Button is one with one background image for each of the three states (normal/hovered/pressed). Just check the constructors to see it. Use the Button.Description class to use up to nine images for a much better looking and scalable Button Widget. Then we give our Button a user-object, which can be evaluated in the onButtonClicked() event. And add a ButtonListener to it. Our HUDExample class implements this interface, so we can take *this* (line 25). The last step to do is to add the new Button Widget to the HUD at some position (line 26). If you want to calculate this position e.g. to center the Button on the HUD, don't take hud.getWidth(), hud.getHeight() nor hud.getSize() for this calculation, but hud.getResolution() or even better hud.getResX() and getResY().

The onButtonClicked(Button, Object) method of ButtonListener is called, when the Button was clicked. There we compare the userObject or the Button instance itself (maybe we have several Buttons) and do the appropriate action (exit the application).

That's it.

The most important Widget types are already implemented. If you decide to implement a new Widget, please share your code. We will love to add it to the toolkit.

There are the following Widgets available at this time:

(in org.xith3d.ui.hud.widgets)

Image, Label, TextField, Button, ToggleButton, Checkbox, RadioButton, ProgressBar, Slider, Scrollbar, ScrollPane, List, ComboBox, Panel, Frame, Dialog

Additionally in org.xith3d.ui.hud.widgets.assemblies.* there're some Widgets assembled of other Widgets. Currently there're FPSCounter (useful to display the current FPS on your HUD) and LoadingScreen (useful to display game loading progress).

Chapter 14b – More Widgets

So you want some more Widgets described? Here we go.

The ToggleButton Widget:

The ToggleButton Widget works exactly the same way as a Button. The only difference is, that it has `isToggled()` and `setToggled()` methods.

The Image Widget:

The Image widget is one of the most simple and basic Widgets. It has a size and a Texture, which you can pass as Texture instance or as a pair of texture resource name and alpha-boolean. The Texture itself is then loaded through the TextureLoader.

```
01 Image image = new Image( 128f, 128f, "precision.png", false );
02 hud.addWidget( image, 100f, 200f );
```

The Label Widget:

The Label Widget is the most important TextWidget implementation and is backed by a TextRectangle (to be found in `org.xith3d.geometry`). So it is a Texture with text rendered on it. If you change the text again and again, the whole Texture needs to be rebuilt. In this case you should consider to use a DynamicLabel Widget, which consists of single letter textures. The Label Widget implements the BorderSettable and PaddingSettable interfaces and can therefore get a Border and padding. See the various Border implementations, which you can pass to the `setBorder()` method. A Label can get a font-color, a font and an alignment. Additionally the Label class implements BackgroundSettableWidget interface, and can therefore get a background Texture. By default it has no background Texture.

```
01 Label label = new Label( 128f, 16f, "Some text\nin two lines" );
02 label.setAlignment( TextAlignment.TOP_LEFT );
03 label.setFontColor( Color3f.GREEN );
04 hud.addWidget( label, 10f, 20f );
```

The TextField Widget:

The TextField Widget is another TextWidget implementation, this time for single line text input. It is defined exactly the same way as a Label (since it extends Label). If the TextField has the focus, a blinking caret is displayed.

```
01 TextField text = new TextField( 128f, 23f, "Some editable text" );
02 hud.addWidget( text, 10f, 50f );
```

The Checkbox Widget:

The Checkbox Widget is another TextWidget implementation. Next to the text a simple checkbox is displayed, which's state can be changed by clicking with the mouse. You can listen for state changes by adding an appropriate Listener.

```
01 Checkbox check = new Checkbox( 128f, 16f, "Some text" );
02 hud.addWidget( check, 10f, 70f );
```

The RadioButton Widget:

The RadioButton Widget is another TextWidget implementation. It is very similar to the Checkbox Widget. The only two differences are, that instead of a checkbox a radiobutton is displayed next to the text and You can add RadioButtons to a ButtonGroup, so that only one member of that group can have the active state. You can listen for state changes by adding an appropriate Listener.

```
01 ButtonGroup stateButtons = new ButtonGroup();
02
03 RadioButton radio1 = new RadioButton( 128f, 16f, "Option 1" );
04 hud.addWidget( radio1, 10f, 90f );
05 stateButtons.addStateButton( radio1 );
06 RadioButton radio2 = new RadioButton( 128f, 16f, "Option 2" );
07 hud.addWidget( radio2, 10f, 110f );
08 stateButtons.addStateButton( radio2 );
```

The Scrollbar Widget:

The The Scrollbar Widget is nothing more than the names lets you gess. You can simply use a ScrollbarListener to get notified when the bar's value changed or link it with a WidgetContainer implementation like Panel.

```
01 Scrollbar scroll1 = new Scrollbar( 128f, Scrollbar.Direction.VERTICAL );
02 scroll1.setLower( 0 );
03 scroll1.setUpper( 100 );
04 scroll1.addScrollbarListener( new MyScrollbarListenerImplementaion() );
05
06 Scrollbar scroll2 = new Scrollbar( 128f, Scrollbar.Direction.HORIZONTAL );
07 scroll2.setLower( 0 );
08 scroll2.setUpper( 100 );
09 scroll2.link( new Panel() );
```

The ScrollPane Widget:

The The ScrollPane Widget is a ContentPaneWrapper. It is rendered around a WidgetContainer (like Panel) and cannot be used without one. It displays two Scrollbars (horizontal and vertical) and any change at the two Scrollbars directly effects the content-pane.

```
01 ScrollPane scrollPane = new ScrollPane( new Panel( 128f, 128f ) );
```

The Slider Widget:

The Slider Widget is used exactly the same as the Scrollbar Widget.

The ProgressBar Widget:

The ProgressBar Widget is a Widget, that displays progress information (e.g. for the game's loading phase). It gets a maximum value and provides some update methods to update progress.

```
01 ProgressBar progress = new ProgressBar( 128f, 64f );
02 progress.setMaxValue( 100 );
03
04 progress.setValue( 1 );
05 progress.setValue( 2 );
06 progress.setValue( 3 );
```

The List Widget:

The List Widget is a Widget, that displays a sequential list of Widgets. It has a size. And if sum of its items' heights gets larger than the List's size, a ScrollBar is used to scroll the items vertically. You can add any Widget type to a List Widget, but most common is a Label List. You need to set the item type by a generic argument. There is a convenience method to add Labels by only invoking the addItem(String) method. This will fail, if the generic argument doesn't extend TextWidget.

```
01 List<Label> list = new List<Label>( 128f, 128f );
02 list.addItem( "Item 1" );
03 list.addItem( "Item 2" );
04 list.addItem( "Item 3" );
05 list.addItem( "Item 4" );
06 hud.addWidget( list, 500f, 50f );
```

The ComboBox Widget:

The ComboBox Widget is a Widget composed of an un-editable TextField and a (dropdown) List<Label> Widget. As the List Widget it extends the AbstractList class and can therefore be handled exactly like a List.

The Panel Widget:

The Panel Widget is a very basic WidgetContainer implementation. It is capable of holding a list of other Widgets, which can be widgetContainers again. A Panel implements BorderSettable, PaddingSettable and BackgroundSettableWidget.

```
01 Panel panel = new Panel( 128f, 128f );
02 panel.addWidget( myLabel1, 10f, 10f );
03 panel.addWidget( myImage1, 50f, 30f );
04 hud.addWidget( panel1, 500f, 300f );
```

The Frame Widget:

The Frame Widget is a basic Window extension. It is a ContentPaneWrapper and is therefore built with a WidgetContainer. If the Frame has a title, it is decorated and can be dragged around with the mouse. A Frame is always BorderSettable. A Frame can solely be added to the HUD itself, but not to any other WidgetContainer!

```
01 Frame frame = new Frame( new Panel( 128f, 128f ), "My first Frame" );
02 frame.getConteintPane().addWidget( myLabel1, 10f, 10f );
03 hud.addWidgetCentered( frame );
```

The Dialog Widget:

The Dialog Widget is a Frame extension and can be used exactly the same. The only difference is, that it blocks any input to other Widgets, when it is attached to the HUD and visible.

The Description classes

All Widget classes have inner Description classes, that can be used to give a set of Widgets the same look and feel, that differs from the current HUD theme. They are all used a very similar way. As an example, I will show the basic usage of a Label.Description class.

```
01 Label.Description labelDesc = new Label.Description();
02
03 labelDesc.setFont( new Font( "My first Frame", Font.BOLD, 12 ) );
04 labelDesc.setFontColor( Color3f.BLUE );
05 labelDesc.setAlignment( TextAlignment.CENTER_CENTER );
06
07 Label label1 = new Label( 128f, 16f, "Label 1", labelDesc );
08 Label label2 = new Label( 128f, 16f, "Label 2", labelDesc );
09 Label label3 = new Label( 128f, 16f, "Label 3", labelDesc );
```

Chapter 14c - Theming the HUD

In the very most cases you'll want all your Widgets to have the same look and feel. The HUD has built in theming support. You have just learned about the Description classes. So the way to a real theme is not too long.

If you don't pass any Texture- or [WIDGET_TYPE].Description parameters to the Widget's constructors, the current Theme defines the look and feel of newly created Widgets. You can also use the getters of the WidgetTheme class to retrieve the [WIDGET_TYPE].Description instances and modify them. They're returned as a clone.

The current Theme can be switched by invoking the static method `setTheme()` of the HUD class, which is overloaded to take either a String or an instance of WidgetTheme. The String version takes the WidgetTheme's name and is only for built-in or once registered themes. To register a new Theme use the `setTheme(WidgetTheme)` method.

Currently there is only one built-in theme in Xith3D, which naturally is the default. It is a GTK like WidgetTheme.

Use the `setTheme(WidgetTheme)` method in the following way:

```
01 HUD.setTheme( new WidgetTheme( new FileInputStream( "my_widget_theme.xwt" ) ) );
```

To create a new Theme you may use the file `GTK.xwt` from the `hud-themes.jar` as an example or template. It is a zip archive with an `.xwt` extension which stands for **X**ith3D**W**idget**T**heme. Copy the file and open the duplicate in your favorite archive tool. It contains a properties file, which is well commented and should be self explanatory and a folder structure holding all the image files in PNG format. This folder structure is fixed in the WidgetTheme class. If you wish to use a different structure, just inherit the WidgetTheme class and use your different structure.

Chapter 15 - Swing integration

But what, if I want to integrate a 3D rendering into my existing Swing application? Now this is really easy. There is a class called Canvas3DPanel which extends the Panel class of the AWT package. So everything you'll have to do besides creating a RenderLoop and all the things you want to have in your 3D world is the following:

```
01 public class EmptyScene
02 {
03     public EmptyScene()
04     {
05         ...
06
07         JPanel myPanel = new JPanel( new GridLayout( 1, 1 ) );
08         Xith3DEnvironment env ...
09
10         Canvas3DPanel c3dPanel = new Canvas3DPanel();
11         myPanel.add( c3dPanel, null );
12
13         env.addCanvas( c3dPanel );
14
15         ...
16
17         myRenderLoop.begin();
18     }
19 }
```

This assumes, you have a working Swing environment and a component like a JPanel with a LayoutManager allowing to stretch the single contained component over the whole area like GridLayout(1, 1) does.

Just create an instance of Canvas3DPanel (see the other constructors, too) and add it to the JPanel like any other Swing or AWT component.

Then add this Canvas3DPanel to the Xith3DEnvironment just like you would do for a normal Canvas3DWrapper or Canvas3D.



Magicosm used Swing for its GUI

Epilogue

Are you completely lost? Maybe your grandma can help you with this clunky NotEnoughTimeException. But for help with Xith3D, you'd better visit the Xith3D forums: <http://www.javagaming.org/forums/index.php?board=30.0>.

If you are in 2060 and the URL is no longer valid, then Google's your friend: <http://www.google.com/search?q=Xith3D>.

We strongly exhort you to take a look at the source. Source code is usually provided with each release, in the src/ directory. Otherwise, if you want the very latest version, you can access to the CVS (more infos on <http://www.xith.org/>)

Good luck with your game/project using Xith3D, and remember, everything's possible !

If you have some flames/wishes/rants/critics about Xith3D or this book, post on our forums. We'll happily ignore flames, fulfill wishes, listen to rants, and take critics into account.

Amos Wenger and Marvin Fröhlich, 4th October 2006