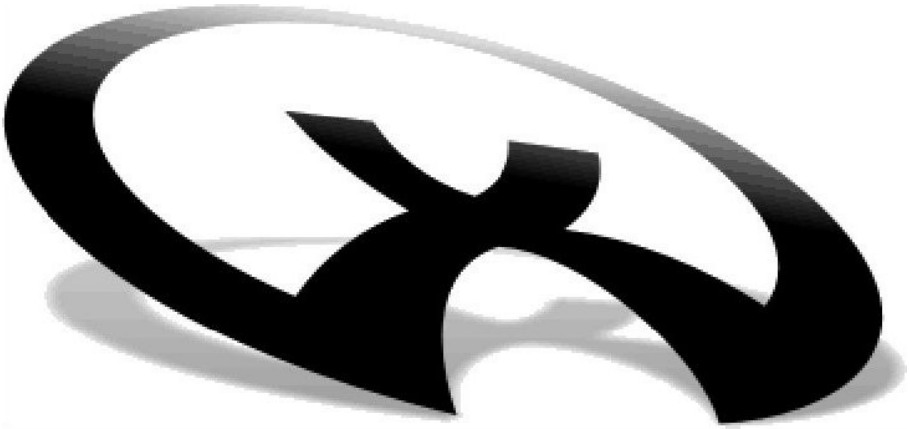
Xith3D in a Nutshell

3rd edition



Authors: *Marvin Fröhlich, Amos Wenger*

Last update: September 1st, 2010

Contents

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 He used to manage the community site, which has been replaced by a new site in 2006, thanks to Amos Wenger.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 VI. 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://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

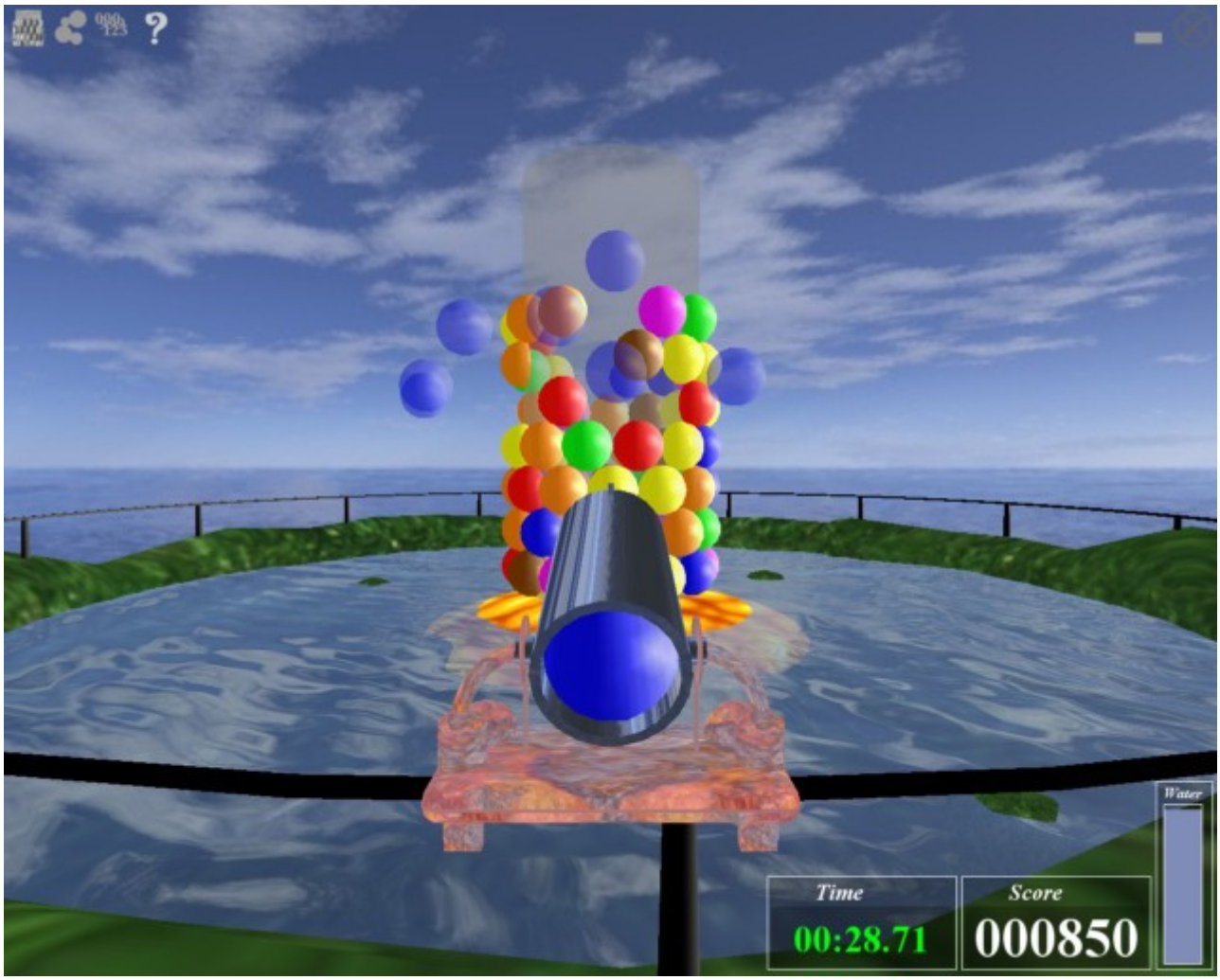
Jack Flowers

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.youtube.com/watch?v=AlsH_AH7CDs

Zplax!

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



http://www.alistairdickie.com/zplax/

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 ) throws Exception
09     {
10         new EmptyScene();
11     }
12 }
```

You can download the full source code of all examples here:
http://xith.org/downloads/docs/xin/xin-3rd-edition-samples.zip

Chapter 1a - Our first empty scene

Setting up an empty Scene is really easy. In prior versions of Xith3D this was done a very similar way as in Java3D and there were some wrapper classes to simplify things up. Now these wrappers have been integrated with the Xith3D API and are the way to go.

Be sure to have a recent Xith3D version. This document matches the API of Xith3D v0.9.7-dev (build 1824).

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

```
01 public class Chapter01a
02 {
03     public Chapter01a()
04     {
05         Xith3DEnvironment env = new Xith3DEnvironment();
06
07         Canvas3D canvas = Canvas3DFactory.createWindowed( 800, 600,
08                                                           "My empty scene" );
09         env.addCanvas( canvas );
10
11         RenderLoop rl = new RenderLoop();
12         rl.setMaxFPS( 120f );
13         rl.setXith3DEnvironment( env );
14
15         // never forget to start the RenderLoop!
16         rl.begin();
17     }
18 }
```

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 Canvas3DFactory, too.

This Canvas3D needs to be added to the environment (line 09), 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 120, 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(float) method like in line 11 and 12.

Now this loop needs to be linked with the the 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 value to the begin() method of RenderLoop telling, whether it is to 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 120 FPS (max).

Chapter 1b - Simplify this code

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

```
01 public class Chapter01b
02 {
03     public Chapter01b()
04     {
05         RenderLoop rl = new RenderLoop( 120f );
06 
07         Xith3DEnvironment env = new Xith3DEnvironment( rl );
08 
09         env.addCanvas( Canvas3DFactory.createWindowed( 800, 600, "My empty scene") );
10         rl.begin();
11     }
12 }
```

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 Chapter01b2 extends RenderLoop
02 {
03     public Chapter01b2()
04     {
05         super( 120f );
06 
07         Xith3DEnvironment env = new Xith3DEnvironment( this );
08 
09         env.addCanvas( Canvas3DFactory.createWindowed( 800, 600, "My empty scene") );
10     }
11 
12     public static void main( String[] args )
13     {
14         Chapter01b2 rl = new Chapter01b2();
15 
16         rl.begin();
17     }
18 }
```

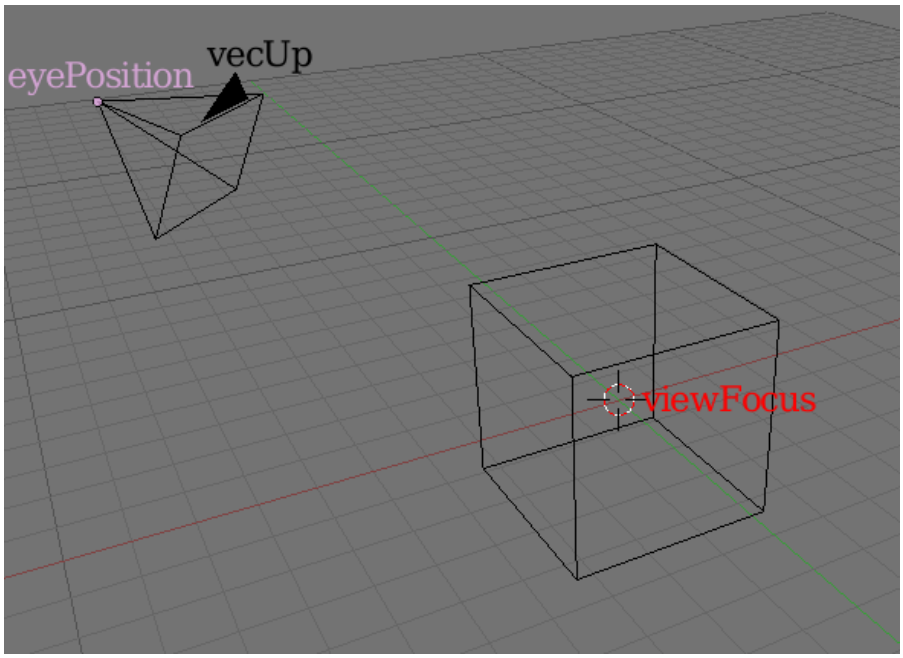
In this case it is good practice to call the begin() method from outside the constructor, like in line #16. All following examples assume so.

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 Chapter02 extends RenderLoop
02 {
03     public Chapter02()
04     {
05         super( 120f );
06 
07         Point3f eyePosition = new Point3f( 0f, 0f, 5f );
08         Point3f viewFocus = Point3f.ZERO;
09         Vector3f vecUp = Vector3f.POSITIVE_Y_AXIS;
10 
11         Xith3DEnvironment env = new Xith3DEnvironment( eyePosition,
12                                                        viewFocus,
13                                                        vecUp,
14                                                        this
15                                                      );
16 
17         // or alternatively...
18         env.getView().lookAt( eyePosition, viewFocus, vecUp );
19 
20         env.addCanvas( Canvas3DFactory.createWindowed( 800, 600, "My empty scene") );
21     }
22 }
```

Note: The parameters chosen for eyePosition, viewFocus and vecUp are the ones used as default by Xith3DEnvironment/View 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 InputAdapterRenderLoop. So we want to enable the user to quit the application by pressing the ESCAPE key or the right mouse button. Too easy...

```
01 public class Chapter03 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     @Override
15     public void onMouseButtonPressed( MouseButtonPressedEvent e, MouseButton button )
16     {
17         if ( button == MouseButtons.RIGHT_BUTTON )
18         {
19             this.end();
20         }
21     }
22 
23     public Chapter03() throws Exception
24     {
25         super( 120f );
26 
27         Xith3DEnvironment env = new Xith3DEnvironment( this );
28 
29         Canvas3D canvas = Canvas3DFactory.createWindowed( 800, 600,
30                                                           "My empty scene" );
31         env.addCanvas( canvas );
32 
33         InputSystem.getInstance().registerNewKeyboardAndMouse( canvas.getPeer() );
34     }
35 }
```

The two new methods reside as empty stups in the InputAdapterRenderLoop 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 #33.

Another way would be to add KeyboardListener, MouseListener or InputListener to your InputSystem or Mouse and Keyboard instances, which can be retrieved from the InputSystem instance.

Please refer to JAGaToo's InputSystem's documentation at (http://sourceforge.net/projects/jagatoo/files/) for further documentation of the input classes.

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 Chapter04 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     @Override
15     public void onMouseButtonPressed( MouseButtonPressedEvent e, MouseButton button )
16     {
17         if ( button == MouseButtons.RIGHT_BUTTON )
18         {
19             this.end();
20         }
21     }
22 
23     public Chapter04() throws Exception
24     {
25         super(); // Note: NO FPS limitation!
26 
27         Xith3DEnvironment env = new Xith3DEnvironment( this );
28 
29         Canvas3D canvas = Canvas3DFactory.createWindowed( 800, 600,
30                                                           "My empty scene" );
31         env.addCanvas( canvas );
32 
33         InputSystem.getInstance().registerNewKeyboardAndMouse( canvas.getPeer() );
34 
35         // Add a listener for frames-per-second (FPS),
36         // that prints the value to the canvas-title.
37         this.addFPSListener( new CanvasFPSListener( canvas ) );
38     }
39 }
```

All FPSListeners (just as our CanvasFPSListener) are invoked every half second by default, 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 Chapter05a 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()
15     {
16         Cube cube = new Cube( 3.0f );
17 
18         Texture tex = TextureLoader.getInstance().getTexture( "stone.jpg" );
19 
20         Appearance app = new Appearance();
21         app.setTexture( tex );
22         cube.setAppearance( app );
23 
24         BranchGroup bg = new BranchGroup();
25         bg.addChild( cube );
26 
27         return( bg );
28     }
29 
30     public Chapter05a() throws Exception
31     {
32         super( 120f );
33 
34         Xith3DEnvironment env = new Xith3DEnvironment( this );
35 
36         Canvas3D canvas = Canvas3DFactory.createWindowed( 800, 600,
37                                                           "The first Shape" );
38         env.addCanvas( canvas );
39 
40         InputSystem.getInstance().registerNewKeyboardAndMouse( canvas.getPeer() );
41 
42         ResourceLocator resLoc = ResourceLocator.create( "test-resources/" );
43         resLoc.createAndAddTSL( "textures" );
44 
45         env.addPerspectiveBranch( createScene() );
46     }
47 }
```

In line #43 we tell the singleton instance of TextureLoader to take “test-resources/textures” 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.scenegraph.primitives directly take the Texture reference, which enables us to simplify the createScene() method a

little:

```
01 public class Chapter05b 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()
15     {
16         Cube cube = new Cube( 3.0f, "stone.jpg" );
17 
18         BranchGroup bg = new BranchGroup();
19         bg.addChild( cube );
20 
21         return( bg );
22     }
23 
24     public Chapter05b() throws Exception
25     {
26         super( 120f );
27 
28         Xith3DEnvironment env = new Xith3DEnvironment( this );
29 
30         Canvas3D canvas = Canvas3DFactory.createWindowed( 800, 600,
31                                                           "The first Shape" );
32         env.addCanvas( canvas );
33 
34         InputSystem.getInstance().registerNewKeyboardAndMouse( canvas.getPeer() );
35 
36         ResourceLocator resLoc = ResourceLocator.create( "test-resources/" );
37         resLoc.createAndAddTSL( "textures" );
38 
39         env.addPerspectiveBranch( createScene() );
40     }
41 }
```

Chapter 6 - Lights

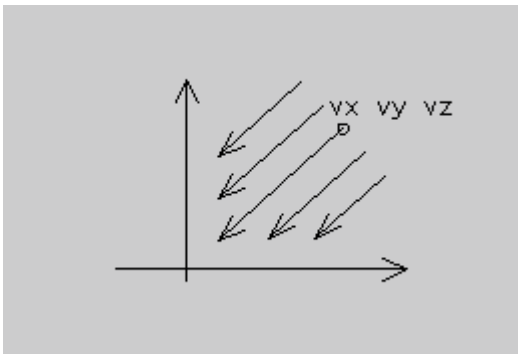
Let there be light! Did you notice, that all shapes had the same, homogeneous light level? This is ok for e.g. a geometry show case program, but for your action-drama game you probably want more. While you can use any (unlimited) number of light sources in your Xith3D scenegraph, only eight (enabled) lights will affect a Shape3D at a time (due to OpenGL limitations).

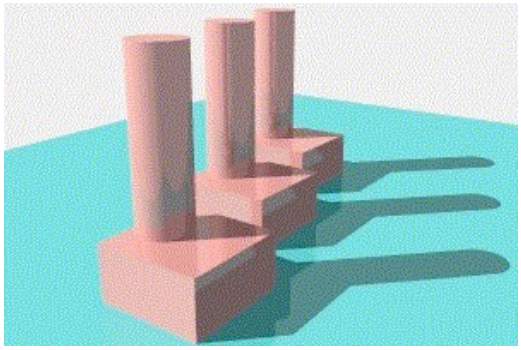
Light sources are set up by adding an instance of Light (actually a subtype) somewhere into the scenegraph. Any Shape3D node located in the same group node, where the Light node resides (or is a (grand-)child of this group) will be affected by this light source.

Types of lights are:

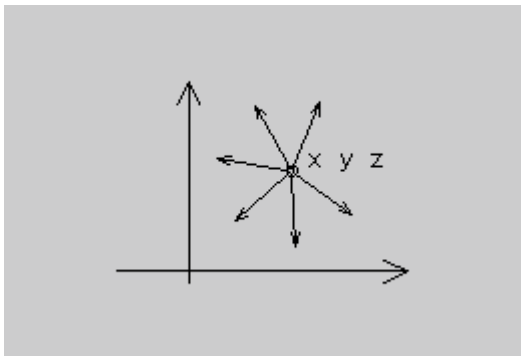
- AmbientLight: It illuminates your whole (subgroup of the) scene in a uniform way. You can adjust the color of the light. It's a good practice to always have a gray (0.3f, 0.3f, 0.3f) light in your game to have everything basely illuminated and you'll see (by mistake) not illuminated shapes.

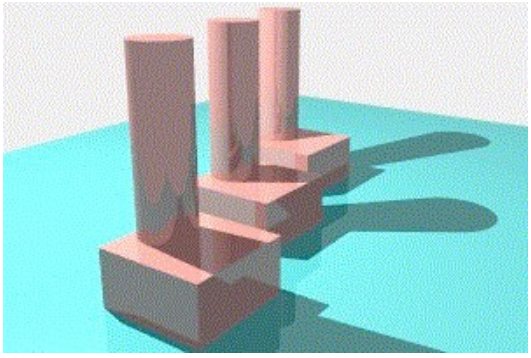
- DirectionalLight: It defines an oriented light source with an origin at infinity. You can adjust the orientation (default is 0f, 0f, -1f) and color.



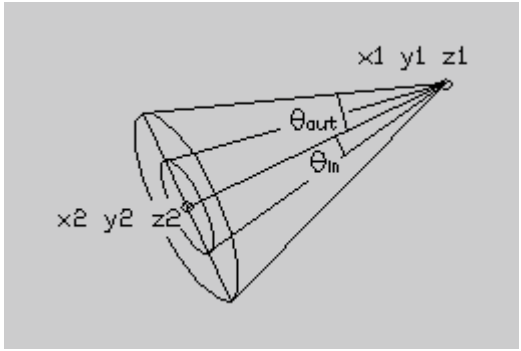


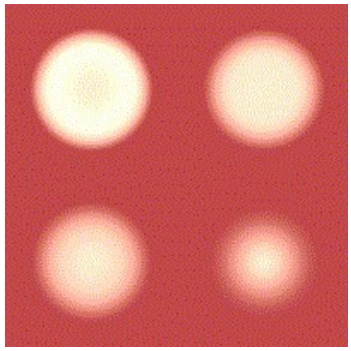
- PointLight: Light emitting point. You can adjust the position (Point3f), attenuation (Tuple3f, values like 0.001f, 0.001f, 0.001f are a good value to start with) and of course color. (Playing with point lights is pure fun.)





- 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), attenuation, spread angle and concentration.





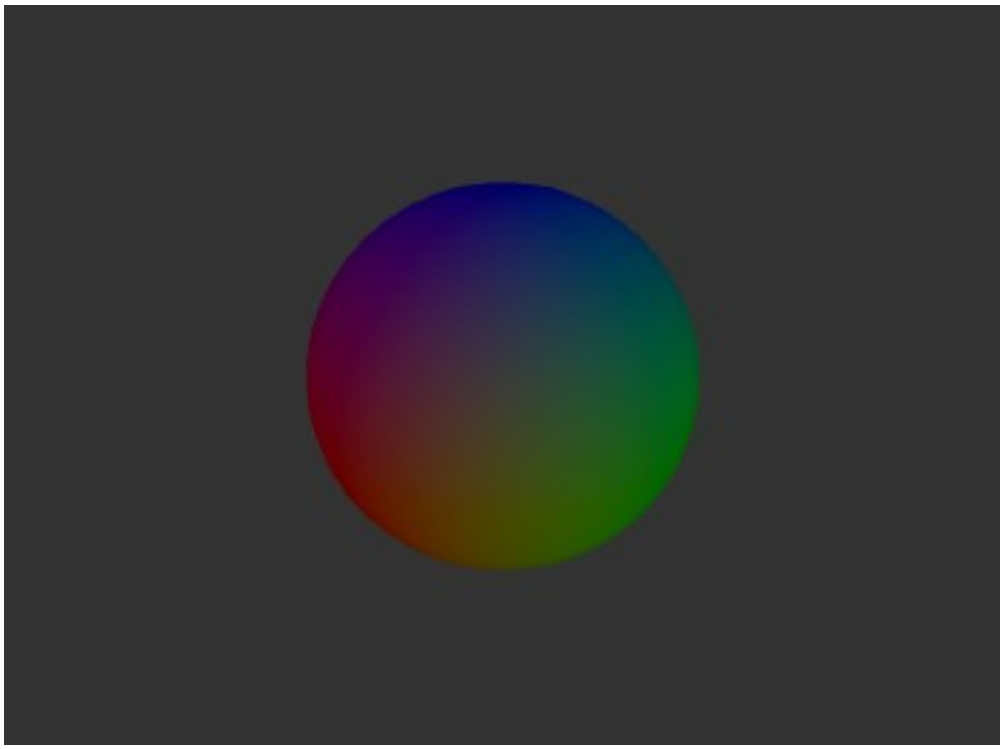
Lights – An example

This 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.

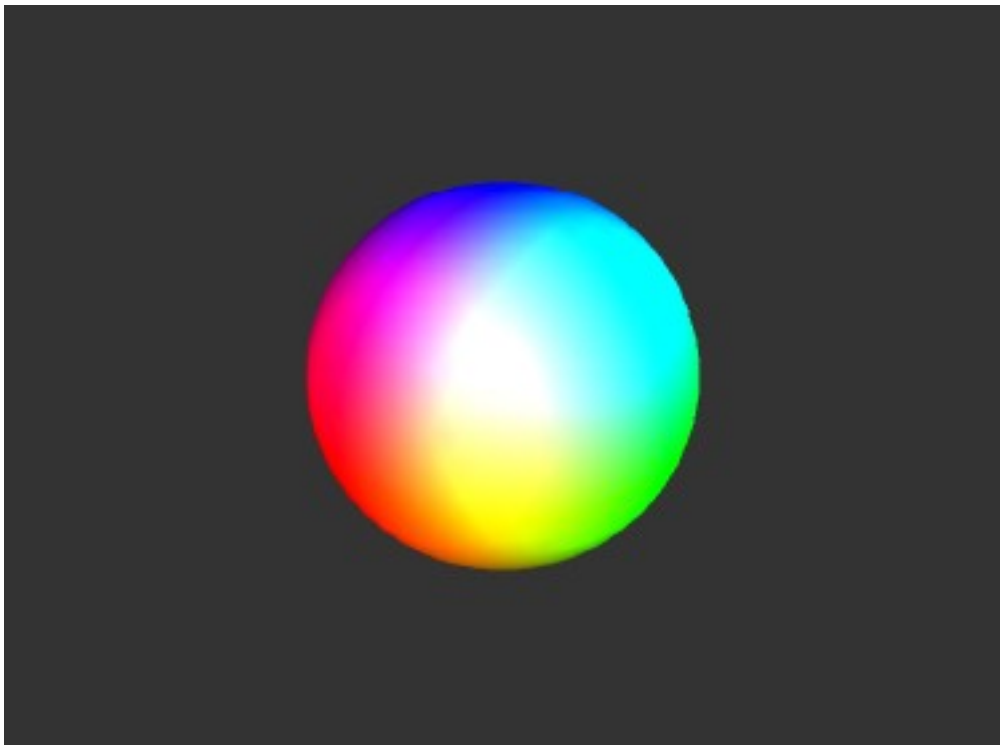
```
01 public class Chapter06 extends InputAdapterRenderLoop
02 {
03     ... // onKeyReleased
04 
05     private BranchGroup createScene()
06     {
07         BranchGroup bg = new BranchGroup();
08 
09         // Add a GeoSphere
10         GeoSphere sph = new GeoSphere( 2.5f, 64, "deathstar.jpg" );
11         bg.addChild( sph );
12 
13         // Adjust appearance
14         Appearance app = sph.getAppearance( true );
15         app.setMaterial( new Material( Colorf.WHITE, Colorf.BLACK,
16                                        Colorf.WHITE, Colorf.BLACK,
17                                        0.8f, Material.AMBIENT, true, true ) );
18 
19         // Add some lighting
20         Light light = new PointLight( Colorf.GREEN,
21                                       new Point3f( 10f, -5f, 5f ),
22                                       new Tuple3f( 0.005f, 0.005f, 0.005f )
23                                     );
24         bg.addChild( light );
25 
26         // To switch off...
27         light.setEnabled( false );
28         // And to switch on...
29         light.setEnabled( true );
30 
31         return( bg );
32     }
33 
34     ... // constructor
35 }
```

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 WHITE.
- Specular color: The RGB specular color of the material (highlights). The range of values is [0f, 1f]. The default specular color is WHITE.
- Emissive color: The RGB color of the light the material emits, if any. The range of values is [0f, 1f]. The default emissive color is BLACK.
- 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)

Remember the cube from the previous chapter. Nice, isn't it? But wouldn't it be even nicer, if the cube was rotating?

```
01 public class Chapter07a extends InputAdapterRenderLoop
02 {
03     private Transform3D t3d;
04     private TransformGroup tg;
05     private AngleInterpolater angleX;
06     private AngleInterpolater angleY;
07
08     ... // onKeyReleased
09
10     private BranchGroup createScene()
11     {
12         Cube cube = new Cube( 3.0f, "stone.jpg" );
13
14         t3d = new Transform3D();
15         tg = new TransformGroup( t3d );
16
17         tg.addChild( cube );
18
19         return( new BranchGroup( tg ) );
20     }
21
22     @Override
23     protected void onRenderLoopStarted()
24     {
25         angleX = new AngleInterpolater( 1.0f );
26         angleY = new AngleInterpolater( 1.0f );
27
28         angleX.startIncreasing( this.getGameMicroTime() );
29         angleY.startIncreasing( this.getGameMicroTime() );
30     }
31
32     @Override
33     protected void prepareNextFrame( long gameTime, long frameTime, TimingMode tm )
34     {
35         super.prepareNextFrame( gameTime, frameTime, tm );
36
37         final long micros = tm.getMicroSeconds( gameTime );
38
39         t3d.rotXYZ( angleX.getValue( micros ), angleY.getValue( micros ), 0.0f );
40         tg.setTransform( t3d );
41     }
42
43     ... // constructor
44 }
```

The methods onRenderLoopStarted() and prepareNextFrame() are again overridden in the RenderLoop class. onRenderLoopStarted() is called before the RenderLoop actually starts working and prepareNextFrame() is called each loop iteration (each frame) right before the actual rendering is performed.

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. Note, that AngleInterpolators always take micro-seconds. So make sure to convert any timing value passed to them beforehand (like in line #37).

Chapter 7b - Easy Rotation

Nice but too complicated? Well we can do easier...

```
01 public class Chapter07b extends InputAdapterRenderLoop
02 {
03     ... // onKeyReleased
04 
05     private BranchGroup createScene( Animator animator )
06     {
07         Cube cube = new Cube( 3.0f, "stone.jpg" );
08 
09         TransformationDirectives rotDirecs =
10                               new TransformationDirectives( 0.3f, 0.2f, 0.0f );
11         RotatableGroup rg = new RotatableGroup( rotDirecs );
12 
13         rg.addChild( cube );
14         animator.addAnimatableObject( rg );
15 
16         return( new BranchGroup( rg ) );
17     }
18 
19     public Chapter07b() throws Exception
20     {
21         super( 120f );
22 
23         Xith3DEnvironment env = new Xith3DEnvironment( this );
24 
25         Canvas3D canvas = Canvas3DFactory.createStandalone( 800, 600,
26                                                             "Animation - Rotation" );
27         env.addCanvas( canvas );
28 
29         InputSystem.getInstance().registerNewKeyboardAndMouse( canvas.getPeer() );
30 
31         ResourceLocator resLoc = ResourceLocator.create( "test-resources/" );
32         resLoc.createAndAddTSL( "textures" );
33 
34         env.addPerspectiveBranch( createScene( this.getAnimator() ) );
35     }
36 }
```

Now this is easy, isn't it? Specially notice line #11, where we register our RotatableGroup, which implements Animatable, as an animatable object to our Animator and immediately start the rotation. Our RenderLoop provides an Animator by default through the getAnimator() method.

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 Chapter07c extends InputAdapterRenderLoop
02 {
03     ... // onKeyReleased
04 
05     private BranchGroup createScene( Animator animator )
06     {
07         Cube cube = new Cube( 3.0f, "stone.jpg" );
08 
09         TransformGroup tg = new TransformGroup();
10         tg.addChild( cube );
11 
12         TransformationDirectives rotDirecs =
13                               new TransformationDirectives( 0.3f, 0.2f, 0f );
14         GroupRotator gr = new GroupRotator( tg, rotDirecs );
15 
16         animator.addAnimatableObject( gr );
17 
18         return( new BranchGroup( tg ) );
19     }
20 
21     ... // constructor
22 }
```

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 update( long gameTime, long frameTime, TimingMode timingMode )
21     {
22         // your code for the operation
23     }
24 }
```

The update method will be called by the render thread each loop iteration before the rendering is performed (from within the prepareNextFrame() method) 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 on the next loop iteration.

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

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

```
01 public class Chapter08a extends RenderLoop
02 {
03     ... // onKeyReleased, createScene
04
05     public Chapter08a() throws Exception
06     {
07         super( 120f );
08
09         ...
10
11         this.getOperationScheduler().scheduleOperation( new MyOperation() );
12     }
13 }
```

Chapter 8b - Intervals

Maybe you want something to be done periodically. Easy with RenderLoop's OperationScheduler:

```
01 public class Chapter08b extends InputAdapterRenderLoop implements IntervalListener
02 {
03     private Appearance app;
04     private Texture[] textures;
05     private int currTex = 0;
06
07     ... // onKeyReleased
08
09     private BranchGroup createScene()
10     {
11         Cube cube = new Cube( 3.0f );
12
13         textures = new Texture[] {
14                     TextureLoader.getInstance().getTexture( "stone.jpg" ),
15                     TextureLoader.getInstance().getTexture( "rustycan.jpg" )
16                                 };
17
18         app = new Appearance();
19         app.setTexture( textures[ currTex ] );
20         cube.setAppearance( app );
21
22         return ( new BranchGroup( cube ) );
23     }
24
25     public void onIntervalHit( Interval interval, long gt, long ft, TimingMode tm )
26     {
27         if ( interval.getName().equals( "my interval" ) )
28         {
29             app.setTexture( textures[ ++currTex % 2 ] );
30             // call interval.kill() to stop and remove the Interval
31         }
32     }
33
34     public Chapter08b() throws Exception
35     {
36         super( 120f );
37
38         Xith3DEnvironment env = new Xith3DEnvironment( this );
39
40         ...
41
42         env.addPerspectiveBranch( createScene() );
43
44         this.getOperationScheduler().addInterval(
45                                         new Interval( 3000000L, "my interval" ) );
46         this.getOperationScheduler().addIntervalListener( this );
47     }
48 }
```

We create an instance of Interval (line 45) which is hit every three seconds (3000000 microseconds) and has “my interval” as it's name.

The method onIntervalHit() is implemented from the IntervalListener interface 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 #27 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 (preferred way)

So you want to pick the cube with your mouse? There are two ways to go. The preferred way is to use the PickingLibrary.

```
01 public class Chapter09a extends InputAdapterRenderLoop implements NearestPickListener
02 {
03     private GroupNode pickGroup;
04     private Canvas3D canvas;
05
06     public void onObjectPicked( PickResult nearest, Object userObject, long pickTime)
07     {
08         System.out.println( "You picked a shape called " +
09                             "\"" + nearest.getNode().getName() + "\"." );
10     }
11
12     public void onPickingMissed( Object userObject, long pickTime );
13     {
14         System.out.println( "You just picked nothing!" );
15     }
16
17     public boolean testIntersectionsInWorldSpaceForPicking()
18     {
19         return ( false );
20     }
21
22
23     @Override
24     public void onMouseButtonPressed( MouseButtonPressedEvent e, MouseButton button )
25     {
26         PickingLibrary.pickNearest( pickGroup, canvas,button,e.getX(),e.getY(),this);
27     }
28
29     ... // onKeyReleased
30
31     private BranchGroup createScene( Animator animator )
32     {
33         Cube cube = new Cube( 3.0f, "stone.jpg" );
34         cube.setName( "my rotating cube" );
35
36         ...
37     }
38
39     public Chapter09a() throws Exception
40     {
41         super( 120f );
42
43         Xith3DEnvironment env = new Xith3DEnvironment( this );
44         Canvas3D canvas = Canvas3DFactory.createWindowed( 800, 600,
45                                                           "Scheduled Picking" );
46         env.addCanvas( canvas );
47         this.canvas = canvas;
48
49         ...
50
51         this.pickGroup = env.addPerspectiveBranch(
52                             createScene( this.getAnimator() ) ).getBranchGroup();
53     }
54 }
```

Isn't this easy? We do the picking on the PickingLibrary. When the picking is done (thread safely), one of the listener methods is called. If the picking is not successful the onPickingMissed() method is called when picking is complete.

Chapter 9b - Alternative Picking

The second way of picking is done with the help of Canvas3D, which simply does GLSelect picking. It works through the same listeners, but is not as feature-rich and can be slower.

```
01 public class Chapter09b extends InputAdapterRenderLoop
02 {
03     private GroupNode pickGroup;
04     private PickEngine pickEngine;
05
06     public void onMouseButtonPressed( MouseButtonPressedEvent e, MouseButton button )
07     {
08         pickEngine.pickNearest( pickGroup, button, e.getX(), e.getY(), this );
09     }
10
11     public void onObjectPicked( PickResult nearest, Object userObject, long pickTime)
12     {
13          System.out.println( "You picked " + nearest.getNode().getName() );
14     }
15
16     public void onPickingMissed( Object userObject, long pickTime )
17     {
18         System.out.println( "You just picked nothing!" );
19     }
20
21     ... // onKeyReleased
22
23     private BranchGroup createScene( Animator animator )
24     {
25         Cube cube = new Cube( 3.0f, "stone.jpg" );
26         cube.setName( "my rotating cube" );
27
28         TransformationDirectives rotDirecs =
29                              new TransformationDirectives( 0.3f, 0.2f, 0f );
30         RotatableGroup rg = new RotatableGroup( rotDirecs );
31
32         rg.addChild( cube );
33         animator.addAnimatableObject( rg );
34
35         this.pickGroup = rg;
36
37         return ( new BranchGroup( rg ) );
38     }
39
40     public Chapter09b() throws Exception
41     {
42         super( 120f );
43
44         Xith3DEnvironment env = new Xith3DEnvironment( this );
45
46         Canvas3D canvas = Canvas3DFactory.createWindowed( 800, 600,
47                                                           "GLSelect Picking" );
48         env.addCanvas( canvas );
49         this.pickEngine = canvas;
50
51         ...
52     }
53 }
```

As you can see GLSelect picking is done very similar. But only use it, if PickingLibrary doesn't do what you need.

Chapter 10 - Screenshots

In some cases you will want to take some screen shots of your rendered scene. This is fairly easy and can be done on two ways.

```
01 public class Chapter10 extends InputAdapterRenderLoop
02 {
03     private ScreenshotEngine shotEngn1;
04     private ScreenshotEngine shotEngn2;
05 
06     public void onKeyReleased( KeyReleasedEvent e, Key key )
07     {
08         switch ( key.getKeyID() )
09         {
10             case F1:
11                 shotEngn1.takeScreenshot( false );
12                 break;
13 
14             case F2:
15                 shotEngn2.takeScreenshot( false );
16                 break;
17 
18             case ESCAPE:
19                 this.end();
20                 break;
21         }
22     }
23 
24     private BranchGroup createScene( Animator animator )
25     {
26         ...
27     }
28 
29     public Chapter10() throws Exception
30     {
31         super( 120f );
32 
33         Xith3DEnvironment env = new Xith3DEnvironment( this );
34 
35         Canvas3D canvas = Canvas3DFactory.createWindowed( 800, 600,
36                                                           "Screenshots" );
37         env.addCanvas( canvas );
38 
39         InputSystem.getInstance().registerNewKeyboardAndMouse( canvas.getPeer() );
40 
41         ResourceLocator resLoc = ResourceLocator.create( "test-resources/" );
42         resLoc.createAndAddTSL( "textures" );
43 
44         env.addPerspectiveBranch( createScene( this.getAnimator() ) );
45 
46         this.shotEngn1 = env.getScreenshotEngine();
47         this.shotEngn2 = canvas;
48     }
49 }
```

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.html, for example.

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

Name	*Extension*	*Texture*	*Frame anim*	*Skeletal anim*	*Supported ?*
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	.md3	x	x		Yes
Doom 3	.md5	X	X	X	Yes
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: Collada, Cal3D or MD5.

Now what are 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.com/. 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 Jeep 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 Chapter11 extends InputAdapterRenderLoop
02 {
03     ... // onKeyReleased
04
05     private BranchGroup createScene( ResourceLocator resLoc, Animator animator )
06                                                                     throws Exception
07     {
08         Model model = ModelLoader.getInstance().loadModel(
09                                         resLoc.getResource( "models/jeep.obj" ) );
10
11         TransformationDirectives rotDirecs =
12                             new TransformationDirectives( 0.3f, 0.2f, 0f );
13         RotatableGroup rg = new RotatableGroup( rotDirecs );
14
15         rg.addChild( model );
16         animator.addAnimatableObject( rg );
17
18         return ( new BranchGroup( rg ) );
19     }
20
21     ... // constructor
22 }
```

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 – Handling Resources

You might have noticed, that all the loading systems (models, textures, sounds, shaders, etc.) take Files, URLs, InputStreams and Readers. But there's only one way to load from everything and to stay comfortable: URLs. Shouldn't there be a way to simplify the loading part? Yes, there is :-).
And you might want to load all your resources at the program start-up.

There is a system in Xith3D, that allows for that. Use it like this:

```
01 public class Chapter12 extends InputAdapterRenderLoop
02 {
03     public static final String RES_MOD1 = "MODEL1";
04     public static final String RES_MOD2 = "MODEL2";
05     public static final String RES_TEX1 = "TEXTURE1";
06     public static final String RES_TEX15 = "TEXTURE15";
07 
08     ... // onKeyReleased
09 
10     private ResourceBag loadResources() throws Exception
11     {
12         ResourceLocator resLoc = ResourceLocator.create( "test-resources/" );
13         resLoc.useAsSingletonInstance();
14 
15         resLoc.createAndAddTSL( "textures" );
16 
17         ResourceLoader loader = new ResourceLoader( resLoc );
18 
19         loader.addRequest( new ModelResourceRequest("models/jeep.obj", RES_MOD1 ));
20         loader.addRequest( new ModelResourceRequest("models/Archer.obj", RES_MOD2 ));
21 
22         loader.addRequest( new TextureResourceRequest( "stone.jpg", RES_TEX1 ) );
23         loader.addRequest( new TextureResourceRequest( "wood.jpg", RES_TEX15 ) );
24 
25         ResourceBag resBag = loader.loadResources();
26 
27         ResourceBag.setSingletonInstance( resBag );
28 
29         return( resBag );
30     }
31 
32     private BranchGroup createScene()
33     {
34         Texture tex1 = ResourceBag.getInstance().getTexture( RES_TEX1 );
35         Cube cube1 = new Cube( 3.0f , tex1 );
36         StaticTransform.translate( cube1, -2f, 0f, 0f );
37 
38         // There is a shortcut for ResourceBag.getInstance():
39         Texture tex2 = ResBag.getTexture( RES_TEX15 );
40         Cube cube2 = new Cube( 3.0f , tex2 );
41         StaticTransform.translate( cube2, 2f, 0f, 0f );
42 
43         ...
44     }
45 
46     public Chapter12() throws Exception
47     {
48         ...
49 
50         loadResources();
51 
52         ...
53     }
54 }
```

First we define unique names for our resources, that are globally accessible (lines #3-6).

Then we create a ResourceLocator with the folder as the base resource, that we will use as the relative location for further resources. We will make this new ResourceLocator globally accessible (maybe we need it later). this is done in the lines #12 and #13.

To be able to load Textures we need to create a TextureStreamLocator. This is simply done in line #15.

Now we need something to load our resources. In line #17 we create a new ResourceLoader and pass the ResourceLocator to its constructor.

In the lines #19-23 we add some parameterized ResourceRequests to the ResourceLoader. They hold all information needed to properly load all of them.

Finally in line #25 we let the ResourceLoader load all the requested resources in a whole. the loadResources() method also takes an instance of LoadingScreenUpdater to make it visualize the loading progress.

The loadResources() method returns a new ResourceBag filled with all of our loaded Resources accessible through their names. We will make this new ResourceBag globally accessible (line #27).

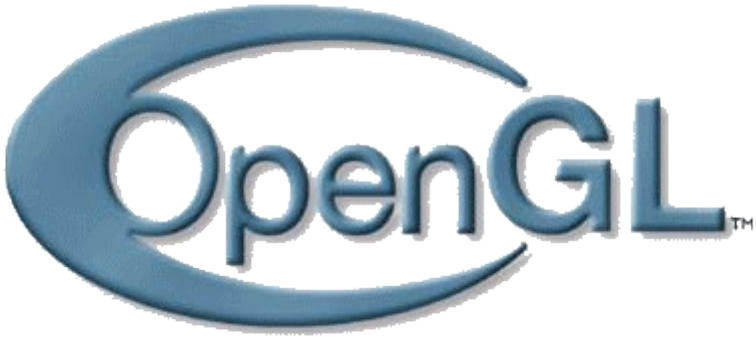
After we have loaded our resources we will use them. In line #34 the normal way is used. But this results in a quite long line of code. And we will use the shortcut instead in line #39, which is absolutely equivalent.

Chapter 13 - 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? It is generally faster (on some systems more on some less) and has guaranteed fullscreen support. The used input bindings work a lot better for games.



But how do I use a specific one? By default, LWJGL is used (JOGL_AWT was the default layer until version 0.9.7-dev build 1824), but it is really easy to switch. Just call a different factory method of Canvas3DFactory. Remember our EmptyScene class. Modify it like this:

```
01 public class Chapter13 extends InputAdapterRenderLoop
02 {
03     ... // onKeyReleased
04
05     public chapter13() throws Exception
06     {
07         super( 120f );
08
09         Xith3DEnvironment env = new Xith3DEnvironment( this );
10
11         env.addCanvas( Canvas3DFactory.createWindowed( OpenGLLayer.JOGL_AWT,
12                                                        800, 600,
13                                                        "Choosing an OpenGLLayer"
14                                                      ) );
15
16        InputSystem.getInstance().registerNewKeyboard( env.getCanvas().getPeer() );
17     }
18 }
```

Chapter 14 – 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 your actual scene (described in chapter 14).

To achieve 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 Chapter14 extends InpuAdapterRenderLoop
02 {
03     ... // onKeyReleased
04
05     private BranchGroup createMainScene()
06     { ... }
07
08     private BranchGroup createParallelScene()
09     { ... }
10
11     private void createSceneGraph( SceneGraph sg )
12     {
13         BaseRenderPassConfig scenePassConfig =
14             new BaseRenderPassConfig( ProjectionPolicy.PERSPECTIVE_PROJECTION );
15         sg.addBranchGraph( createMainScene(), scenePassConfig );
16
17         BaseRenderPassConfig paraPassConfig =
18             new BaseRenderPassConfig( ProjectionPolicy.PARALLEL_PROJECTION );
19         sg.addBranchGraph( createParallelScene(), paraPassConfig );
20     }
21
22     public Chapter14() throws Exception
23     {
24         super( 120f );
25
26         Xith3DEnvironment env = new Xith3DEnvironment( this );
27
28         ...
29
30         createSceneGraph( env );
31     }
32 }
```

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 simplify this coding:

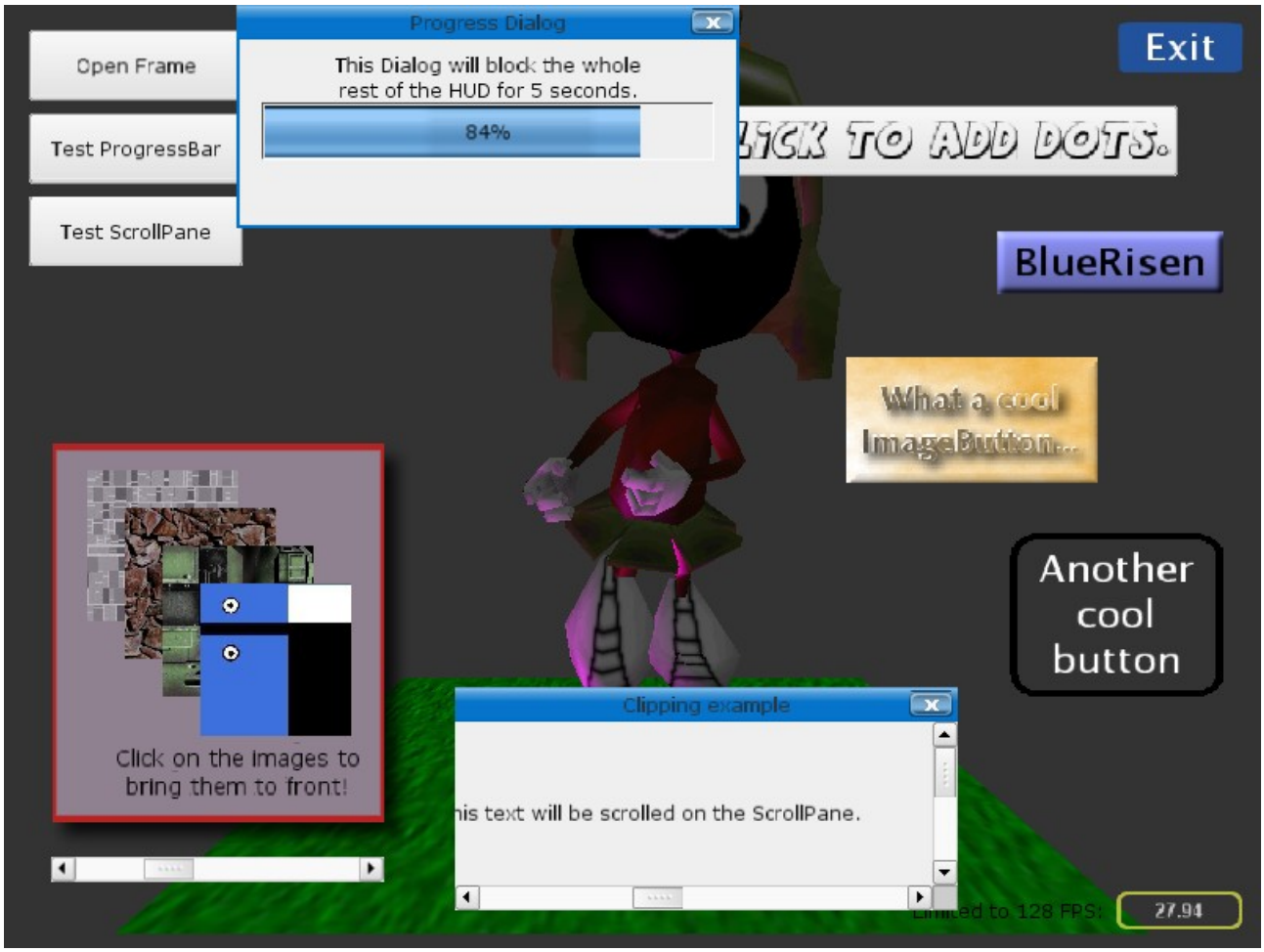
```
01         env.addPerspectiveBranch( createMainScene() );
02         env.addParallelBranch( createParallelScene() );
```

Have a look at the BaseRenderPassConfig class, that is attached to each RenderPass instance. It offers several settings to be adjusted, that might be useful for you.

Chapter 15a - 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 Chapter15a extends InputAdapterRenderLoop implements ButtonListener
02 {
03     ... // onKeyReleased
04 
05     public void onButtonPressed( Button button, Object userObject )
06     {
07         if ( userObject.equals( "EXIT_BUTTON" ) )
08         {
09             this.end();
10         }
11     }
12 
13     private BranchGroup createMainScene()
14     {
15         ...
16     }
17 
18     private HUD createHUD( Sized2iRO canvasSize )
19     {
20         HUD hud = new HUD( canvasSize, 800f );
21 
22         Button button = new Button( 200f, 60f, "Click me to exit" );
23         button.setUserObject( "EXIT_BUTTON" );
24         button.addButtonListener( this );
25         hud.getContentPane().addWidgetCentered( button );
26 
27         return( hud );
28     }
29 
30     public Chapter15a() throws Exception
31     {
32         super( 120f );
33 
34         Xith3DEnvironment env = new Xith3DEnvironment( this );
35 
36         Canvas3D canvas = Canvas3DFactory.createWindowed( 800, 600, "HUD example" );
37         env.addCanvas( canvas );
38 
39         InputSystem.getInstance().registerNewKeyboard( canvas.getPeer() );
40 
41         ResourceLocator resLoc = ResourceLocator.create( "test-resources/" );
42         resLoc.createAndAddTSL( "textures" );
43 
44         env.addPerspectiveGraph( createMainScene() );
45 
46         env.addHUD( createHUD( canvas ) );
47     }
48 }
```

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

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 of the Canvas3D. We only need to pass the horizontal (virtual) resolution to the HUD constructor and the vertical is automatically calculated.

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 Chapter15a class implements this interface, so we can take *this* (line 24). The last step to do is to add the new Button Widget to the HUD at some position (line 25). 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)
Button, Checkbox, ComboBox, Dialog, FPSCounter, Frame, Image, Label, List, Panel, ProgressBar, QuadDivider, RadioButton, Scrollbar, ScrollPanel, Slider, TextField, ToggleButton, Widget3D

Additionally in org.xith3d.ui.hud.widgets.assemblies there are some Widgets assembled of other Widgets. Currently there are HUDConsole (like the console available for instance in Quake 3) and LoadingScreen (useful to display game loading progress).

Chapter 15b – 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.getContentPane().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( Colorf.GREEN );
04 hud.getContentPane().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.getContentPane().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.

```
Checkbox check = new Checkbox( 128f, 16f, "Some text" );
hud.getContentPane().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.

```
ButtonGroup stateButtons = new ButtonGroup();

RadioButton radio1 = new RadioButton( 128f, 16f, "Option 1" );
hud.getContentPane().addWidget( radio1, 10f, 90f );
stateButtons.addStateButton( radio1 );
RadioButton radio2 = new RadioButton( 128f, 16f, "Option 2" );
hud.getContentPane().addWidget( radio2, 10f, 110f );
stateButtons.addStateButton( radio2 );
```

The Scrollbar Widget:

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

```
Scrollbar scroll1 = new Scrollbar( 128f, Scrollbar.Direction.VERTICAL );
scroll1.setLower( 0 );
scroll1.setUpper( 100 );
scroll1.addScrollbarListener( new MyScrollbarListenerImplementaion() );

Scrollbar scroll2 = new Scrollbar( 128f, Scrollbar.Direction.HORIZONTAL );
scroll2.setLower( 0 );
scroll2.setUpper( 100 );
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.

```
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.

```
ProgressBar progress = new ProgressBar( 128f, 64f );
progress.setMaxValue( 100 );

progress.setValue( 1 );
progress.setValue( 2 );
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.

```
List<Label> list = new List<Label>( 128f, 128f );
list.addItem( "Item 1" );
list.addItem( "Item 2" );
list.addItem( "Item 3" );
list.addItem( "Item 4" );
hud.getContentPane().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 therefor 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.

```
Panel panel = new Panel( 128f, 128f );
panel.addWidget( myLabel1, 10f, 10f );
panel.addWidget( myImage1, 50f, 30f );
hud.getContentPane().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. It 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.getContentPane().addWidget( myLabel1, 10f, 10f );
03 hud.addWindowCentered( 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( Colorf.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 15c - 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 xith3d.jar as an example or template (in the resources/org/xith3d/hud/themes folder). 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 16 - 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 Chapter16
02 {
03     public Chapter16()
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 environement 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 clunksy NotEnoughTimeException. But for help with Xith3D, you'd better visit the Xith3D forums: http://xith.org/forum.

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 SVN (more infos on http://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.

Marvin Fröhlich and Amos Wenger, September 1st 2010.