Caliko User Guide



Alastair Lansley / Federation University Australia

Contents

[Introduction 3](#_Toc36722631)

[Demo / Quick Start 4](#_Toc36722632)

[Download and Structure 5](#_Toc36722633)

[Installation and Setup 6](#_Toc36722634)

[Visualisation Setup 7](#_Toc36722635)

[Problems running the Caliko Demo on a Mac 9](#_Toc36722636)

[IK Terminology and Workflow 11](#_Toc36722637)

[Structures 12](#_Toc36722638)

[Joints and Restrictions 12](#_Toc36722639)

[Basebone Restrictions 12](#_Toc36722640)

[Joint Restrictions 13](#_Toc36722641)

[Solving IK Chains and Structures 13](#_Toc36722642)

[Vectors and Matrices 14](#_Toc36722643)

[Reproducible Sequences 14](#_Toc36722644)

[Obtaining Pitch, Yaw and Roll Angles for 3D Vectors / Bones 14](#_Toc36722645)

[Usage Examples 15](#_Toc36722646)

[2D Demo 1 - Chain with GLOBAL\_ABSOLUTE basebone constraint and joint constraints 15](#_Toc36722647)

[2D Demo 2 - Chain with fixed base and multiple unconstrained bones 16](#_Toc36722648)

[2D Demo 3 - Chain with fixed base and multiple constrained bones 16](#_Toc36722649)

[2D Demo 4 - Chain with fixed base and multiple constrained bones 17](#_Toc36722650)

[2D Demo 5 - Multiple connected chains with LOCAL\_RELATIVE basebone constraints 18](#_Toc36722651)

[2D Demo 6 - Multiple connected chains with LOCAL\_ABSOLUTE basebone constraints 20](#_Toc36722652)

[2D Demo 7 – Chains with embedded targets 21](#_Toc36722653)

[2D Demo 8 – Multiple Nested Chains in a Semi Random Configuration 22](#_Toc36722654)

[2D Demo 9 - World-Space 2D Constraints 23](#_Toc36722655)

[3D Demo 1 - Unconstrained bones 25](#_Toc36722656)

[3D Demo 2 - Rotor / ball joint constrained bones 25](#_Toc36722657)

[3D Demo 3 - Chains with rotor constrained basebones 26](#_Toc36722658)

[3D Demo 4 - Chains with freely rotating global hinges 28](#_Toc36722659)

[3D Demo 5 - Global hinges with reference axis constraints 29](#_Toc36722660)

[3D Demo 6 - Chains with local hinges 30](#_Toc36722661)

[3D Demo 7 - Connecting chains 31](#_Toc36722662)

[3D Demo 12 – Connected chains with embedded targets 33](#_Toc36722663)

# Introduction

The Caliko library is a free open-source software (FOSS) implementation of the FABRIK (Forward And Backward Reaching Inverse Kinematics) algorithm created by Aristidou and Lasenby. It is written in the Java programming language.

The inverse kinematics (IK) algorithm is implemented in both 2D and 3D, and incorporates a variety of joint constraints, as well as the ability to connect multiple IK chains together in a hierarchy.

The library allows for the simple creation and solving of multiple IK chains as well as visualisation of these solutions. You can watch a short video outlining the setup and functionality of it here: <https://www.youtube.com/watch?v=wEtp4P2ucYk>

The Caliko library is licensed under the MIT software license and the source code is freely available for use and modification at: <https://github.com/feduni/caliko>

Please feel free to report any issues at: <https://github.com/feduni/caliko/issues>

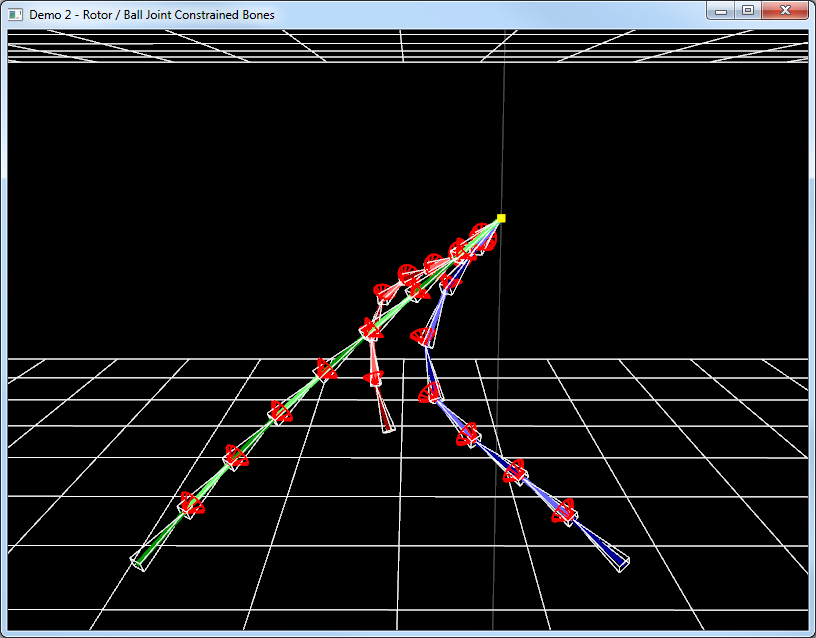
Further details on the FABRIK algorithm itself can be found in the following paper:

Aristidou, A., & Lasenby, J. (2011). FABRIK: a fast, iterative solver for the inverse kinematics problem. Graphical Models, 73(5), 243-260.

# Demo / Quick Start

If you'd like to see what the Caliko library does, then a YouTube demonstration video is available at:

<https://www.youtube.com/watch?v=D9-V66m9DVI>



If you'd like to experiment with the demo yourself then you can either download a pre-compiled release from <https://github.com/FedUni/caliko/releases> or you can use [git](https://git-scm.com/) to clone the Caliko repository at <https://github.com/FedUni/caliko> and build the library via [maven](https://maven.apache.org/) by using the command:

mvn package

Once you have the library, extract the **caliko-distribution-<version>.zip** file in the demo app from the **caliko-distribution/target** folder you can run the 'jar with resources' demo by right-clicking on it and selecting **[Open]** from the pop-up menu (on Windows), or from the command line by issuing the command:

java -jar caliko-<version>-demo-jar-with-dependencies.jar

To run the demo application Java 11 and OpenGL 3.3 or newer are required. The version of Java used can be modified by adjusting the value in the parent (i.e. top level) **pom.xml** file's *<release>* element.

In 2D mode, clicking or holding the left mouse button (LMB) updates the target location to be at the location of the cursor. In 3D mode holding the LMB and moving the mouse allows you to look around. The keyboard controls used in the demo are listed below:

Table 1 - Default controls for the Caliko demonstration application.

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Action | Input | Action |
| Up/Down cursor keys | Toggle 2D/3D mode | C | Toggle drawing constraints |
| Left/Right cursor keys | Previous/Next demo | X | Toggle drawing axes (3D) |
| W/A/S/D keys | Move camera forward/back/left/right (3D) | M | Toggle drawing models (3D) |
| P | Toggle orthographic / perspective projection (3D) | L | Toggle drawing lines |
| F | Toggle fixed-base mode | Space | Toggle pausing moving target (3D) |
| R | Rotate base locations (3D) | Esc | Exit demo |

# Download and Structure

The Caliko library is publicly hosted on GitHub at: <https://github.com/feduni/caliko>

Precompiled releases may be found at: <https://github.com/FedUni/caliko/releases>

Once Caliko has been built with maven as per the above instructions, the **caliko-distribution/target/caliko-distribution-<version>.zip** file will exist with the following structure:

**caliko**

**│ caliko-distribution-1.3.7-sources.zip**

**│ Changelog.txt**

**│ License.txt**

**│**

**├───doc**

**│ caliko-distribution-1.3.7-javadoc.jar**

**│ caliko-user-guide.pdf**

**│**

**└───jar**

**caliko-1.3.7.jar**

**caliko-demo-1.3.7-jar-with-dependencies.jar**

**caliko-demo-1.3.7.jar**

**caliko-visualisation-1.3.7.jar**

Figure 1 - The Caliko release directory structure, specific version numbers may vary by release.

The **caliko-distribution-<version>-sources.zip** file contains all the source code for Caliko, its utils, visualisation and demo packages.

The **doc** folder contains this user guide and a combined archive of all Javadoc documentation for the library's application programming interface (API).

The **jar** folder contains compiled versions of the core / headless Caliko classes, along with the optional visualisation component. Note that you will only be able to directly run the 'jar-with-dependencies' because it contains the LWJGL3 library required by the demos.

# Installation and Setup

If using the jars instead of the source code, then the Caliko library must be added to your build path before it can be used.

In an IDE such as Eclipse this is typically achieved by creating a new User Library, and then pointing at the jar file(s), along with the optional source code and Javadoc. For example:

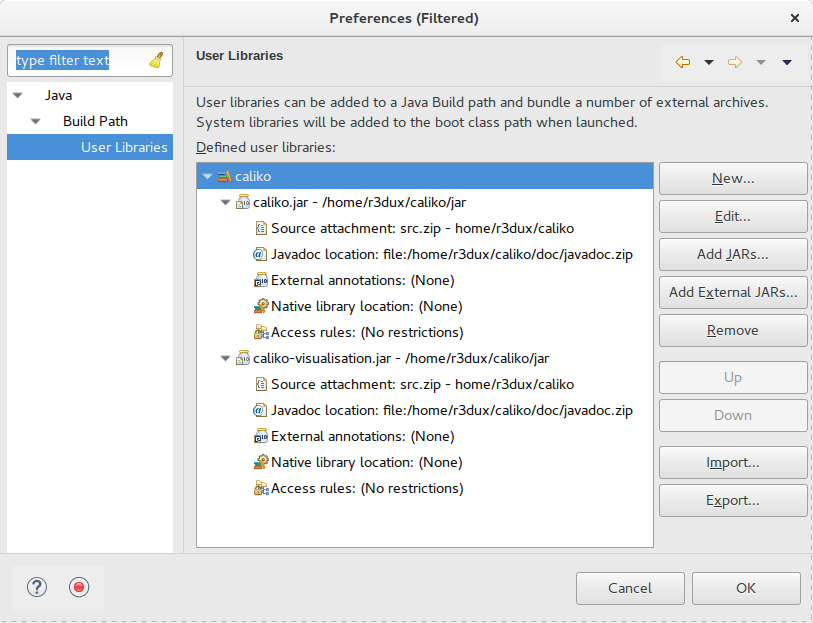


Figure 2 - An example of setting up the Caliko library as a User Library in Eclipse.

To use the visualisation component, which uses the LWJGL3 library, a similar process may be used to create a LWJGL3 user library, as shown below in Figure 3:

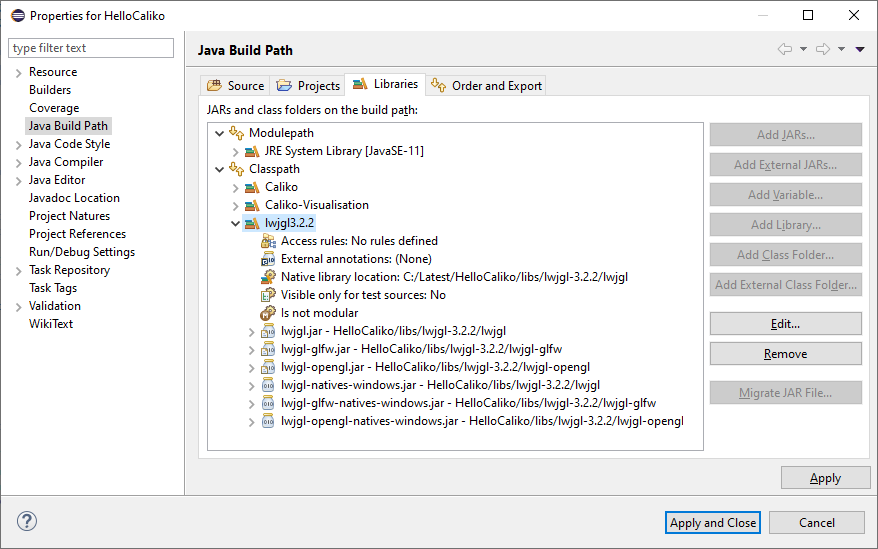


Figure 3 - An example of setting up the LWJGL3 library as a User Library in Eclipse, specifying we want to use the Windows natives.

# Visualisation Setup

The LWJGL3 library guide at <https://www.lwjgl.org/guide> explains how to create an OpenGL context which can be used for drawing Caliko IK chains. The specific version of LWJGL3 in use is presently **3.2.2**. You may wish to look at the Caliko demo app's **au.edu.federation.demo** package, specifically the **OpenGLWindow** and **Application** classes to see how I've pieced this together.

Potentially the simplest setup you could get away for visualisation of IK chains is similar to the below:

**import** org.lwjgl.glfw.\*;

**import** org.lwjgl.opengl.\*;

**import** **static** org.lwjgl.glfw.GLFW.\*;

**import** **static** org.lwjgl.opengl.GL11.\*;

**import** **static** org.lwjgl.system.MemoryUtil.\*;

**import** au.edu.federation.caliko.\*;

**import** au.edu.federation.utils.\*;

**import** au.edu.federation.caliko.visualisation.\*;

**public** **class** HelloCaliko

{

// We need to strongly reference callback instances.

**private** GLFWErrorCallback errorCB;

**private** GLFWKeyCallback keyCB;

**int** WIDTH = 800; **int** HEIGHT = 600; // Window width and height

**private** **long** window; // Window handle

FabrikChain2D chain = **new** FabrikChain2D(); // Create a new 2D chain

// 2D projection matrix. Params: Left, Right, Top, Bottom, Near, Far

Mat4f mvpMatrix = Mat4f.*createOrthographicProjectionMatrix*(

-(**float**)WIDTH/2.0f, (**float**)WIDTH/2.0f,

(**float**)HEIGHT/2.0f, -(**float**)HEIGHT/2.0f,

1.0f, -1.0f );

**public** **void** run()

{

// Create our chain

FabrikBone2D base = **new** FabrikBone2D(**new** Vec2f(), **new** Vec2f(0.0f, 50.0f));

chain.addBone(base);

**for** (**int** boneLoop = 0; boneLoop < 5; ++boneLoop)

{

chain.addConsecutiveBone(**new** Vec2f(0.0f, 1.0f), 50.0f);

}

**try**

{

init();

loop();

}

**finally**

{

// Free the keyboard callback and destroy the window

keyCB.close();

*glfwDestroyWindow*(window);

// Terminate GLFW and free the error callback

*glfwTerminate*();

*glfwSetErrorCallback*(**null**).free();

}

}

**private** **void** init()

{

// Setup an error callback. The default implementation

// will print the error message in System.err.

*glfwSetErrorCallback*(errorCB = GLFWErrorCallback.*createPrint*(System.***err***));

// Initialize GLFW. Most GLFW functions will not work before doing this.

**if** ( !*glfwInit*() )

**throw** **new** IllegalStateException("Unable to initialize GLFW");

*glfwWindowHint*(***GLFW\_RESIZABLE***, ***GLFW\_TRUE***); // the window will be resizable

// Create the window

window = *glfwCreateWindow*(WIDTH, HEIGHT, "Hello, Caliko!", ***NULL***, ***NULL***);

**if** (window == ***NULL***)

**throw** **new** RuntimeException("Failed to create the GLFW window");

// Setup a key callback

*glfwSetKeyCallback*(window, keyCB = **new** GLFWKeyCallback() {

@Override

**public** **void** invoke(**long** window, **int** key, **int** scancode, **int** action, **int** mods)

{

**if** ( key == ***GLFW\_KEY\_ESCAPE*** && action == ***GLFW\_RELEASE*** )

*glfwSetWindowShouldClose*(window, **true**);

}

});

// Get the resolution of the primary monitor

GLFWVidMode vidmode = *glfwGetVideoMode*( *glfwGetPrimaryMonitor*() );

// Center our window

*glfwSetWindowPos*(window, (vidmode.width() - WIDTH) / 2,

(vidmode.height() - HEIGHT) / 2);

*glfwMakeContextCurrent*(window); // Make the OpenGL context current

*glfwSwapInterval*(1); // Enable v-sync

*glfwShowWindow*(window); // Make the window visible

}

**private** **void** loop()

{

// This line is critical for LWJGL's interoperation with GLFW's

// OpenGL context, or any context that is managed externally.

// LWJGL detects the context that is current in the current thread,

// creates the GLCapabilities instance and makes the OpenGL

// bindings available for use.

GL.*createCapabilities*();

*glClearColor*(0.0f, 0.0f, 0.0f, 0.0f); // Set the clear color

Vec2f offset = **new** Vec2f(150.0f, 0.0f);

Vec2f target = **new** Vec2f(100.0f, 100.0f);

// Run the rendering loop until the user has attempted to close

// the window or has pressed the ESCAPE key.

**while** ( *glfwWindowShouldClose*(window) == **false** ) {

*glClear*(***GL\_COLOR\_BUFFER\_BIT*** | ***GL\_DEPTH\_BUFFER\_BIT***); // Clear buffers

chain.solveForTarget( target.plus(offset) ); // Solve the chain

FabrikLine2D.*draw*(chain, 3.0f, mvpMatrix); // Draw the chain

*glfwSwapBuffers*(window); // Swap colour buf.

// Rotate the offset 1 degree per frame

offset = Vec2f.*rotateDegs*(offset, 1.0f);

// Poll for window events. The key callback above will only be

// invoked during this call.

*glfwPollEvents*();

}

}

**public** **static** **void** main(String[] args) { **new** HelloCaliko().run(); }

}

When this code is executed, we'll have a simple 2D chain with six bones (the basebone plus the five we added) which is constantly solved for the given rotating location, as shown below in Figure 4.

## Problems running the Caliko Demo on a Mac

Running the Caliko demo on a Mac may result in a crash on start-up. It has been reported that performing the following modifications mitigate this issue (with thanks to github user *jillemash*):

* Add -XstartOnFirstThread VM arg ( saw here <https://stackoverflow.com/a/56539190> and elsewhere)
* Uncomment Fwd compat hint line (OpenGLWindow:93)
* Comment ShaderProgram:392 to not throw RuntimeException "Could not validate shader program"...

As I don't have any Mac hardware to test on I'm unable to provide a more comprehensive solution at this time. Please see <https://github.com/FedUni/caliko/issues/1> for further details.

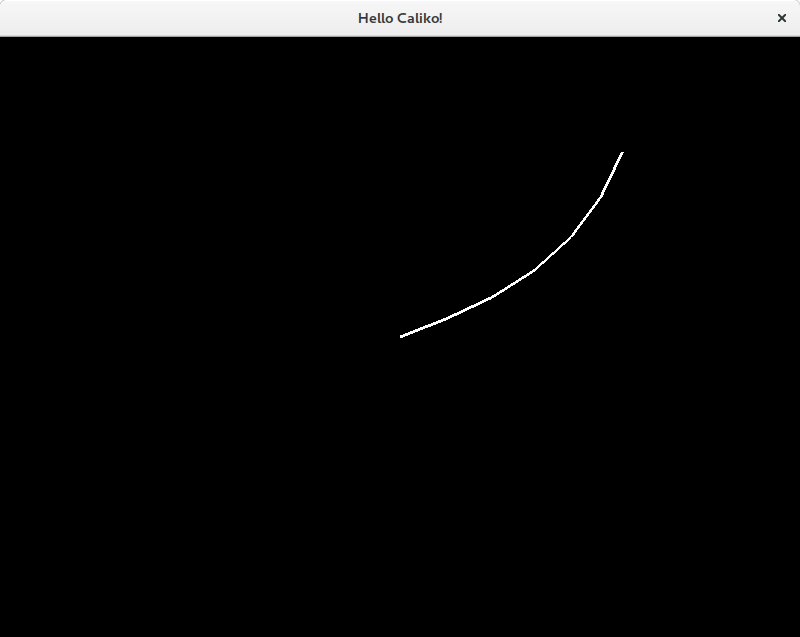


Figure 4 - An example of running the above sample code to create an OpenGL context, and solve and display a simple 2D chain. It's more interesting to look at in motion than via a static screenshot ;-)

# IK Terminology and Workflow

An IK **chain** is a collection of IK **bones** which are connected to each other. The very first bone in the chain is called the ***basebone***, and the start location of the basebone is called the ***base location***.

Each bone has a start location and an end location which, depending on whether we are working in 2D or in 3D, will be defined in either 2D or 3D space. The end location of the final bone in a chain is called the ***end-effector***.

Each bone also has a single ***joint*** which may control the allowable movement of the bone.

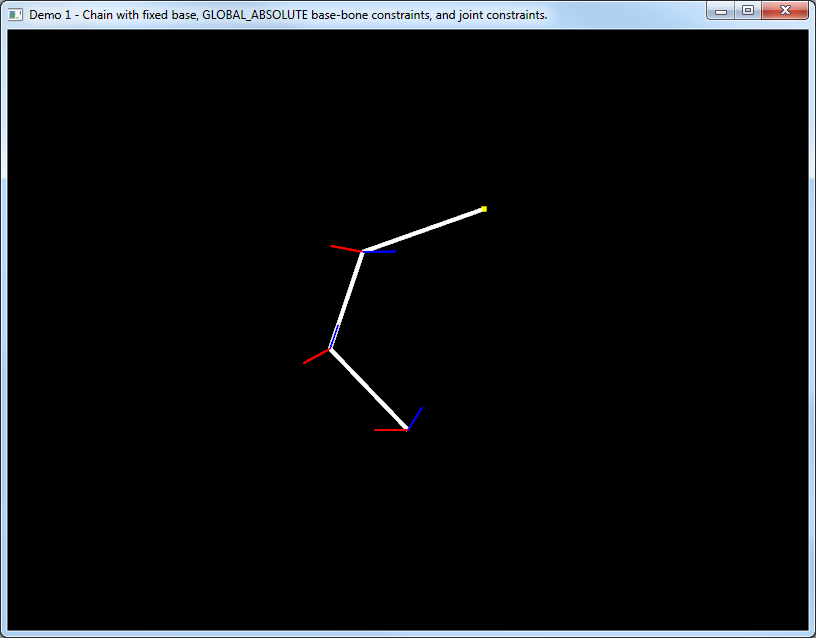
The typical workflow of an IK setup is that:

* A chain is created,
* One or more bones are added to this chain, then
* An attempt is made to **solve** the chain for a given target location.

It is during this solve attempt that, in this case, the FABRIK algorithm is executed which may alter the configuration of the chain so that the end-effector is as close to the target location as possible.

During this solve attempt:

* The length of the bones do not change, and
* The end location of one bone and the start location of the following bone overlap precisely.



End effector is at target - successful solve!

Basebone with fixed base location and global / world-space basebone constraint.

Joint constraints are relative to previous bone.

Figure 5 - A simple IK chain in 2D. It contains three bones, has a fixed base location, and has been successfully solved.

### Structures

In the Caliko library, multiple IK chains may be connected to each other by placing them into a special 'holder' object called a ***structure***. For example, after adding an initial chain to a structure, we might decide to add a second chain to the structure where the basebone of that second chain is connected to the end joint of the third bone in the initial chain (or the start joint of the second bone etc. - it's entirely up to you).

As such, when the first chain moves, the second chain's base location also moves in order to remain attached.

### Joints and Restrictions

Each bone in a chain may be configured to allow only a restricted subset of motion. In the Caliko library there are two specific types of restriction:

* **Basebone restrictions** - which are applied to the ***chain***, and
* **Joint restrictions** - which are applied to individual ***bones***.

Depending on whether you are working in 2D or 3D there are a variety of different basebone and standard joint restriction types.

### Basebone Restrictions

A basebone restriction is a special-case of standard restriction, and is required because the basebone itself is the very first bone in a chain and as such does not have a previous bone which it may be restricted about.

In 2D, the basebone restriction types are:

* **NONE** - No constraint,
* **GLOBAL\_ABSOLUTE** - Constrained about a global / world-space direction,
* **LOCAL\_ABSOLUTE** - Constrained about the direction of the connected bone, and
* **LOCAL\_RELATIVE** - Constrained about a direction relative to that of the connected bone.

The LOCAL\_ABSOLUTE and LOCAL\_RELATIVE basebone constraint types are only available to be used by chains which are connected to other chains.

In 3D, the basebone restriction types are:

* **NONE** - No constraint,
* **GLOBAL\_ROTOR** - Ball-joint constrained about a global / world-space direction,
* **LOCAL\_ROTOR**  - Ball-joint constrained about the direction of the connected bone,
* **GLOBAL\_HINGE -** Hinge constrained about a world-space axis, and
* **LOCAL\_HINGE -** Hinge constrained about an axis relative to that of the connected bone.

As before, the LOCAL\_ROTOR and LOCAL\_HINGE basebone constraint types are only available to be used by chains which are connected to other chains. In addition, hinge constraints may have an additional ***reference axis*** constraint which is the direction within the axis of the hinge about which clockwise/anticlockwise movement is allowed.

To put this in perspective - if you think of the front door of your house rotating on its hinges: its reference axis would be when the door is closed, and it can likely rotate zero degrees one way (i.e. it doesn't open outwards) and maybe up to 100 degrees or such the other way to let people in and out.

3D parabolic constraints may be added in a future release, but are not implemented at the present time.

### Joint Restrictions

As previously mentioned, each bone has a single joint which can be thought of as being at the start of the bone. In 2D there no specific joint types as the bones may only rotate clockwise and anticlockwise, however, 2D joint constraints may be configured to operate in a GLOBAL or LOCAL manner - where a local constraint is relative to the coordinate system of the previous bone in the chain.

In 3D, there are three distinct types of joint, which are:

* **BALL** - A 'rotor' / ball-joint constraint about the previous bone in the chain,
* **GLOBAL\_HINGE** - A world-space hinge constraint, or
* **LOCAL\_HINGE** - A hinge constraint relative to the previous bone in the chain.

The default joint type of a 3D bone is BALL, and the default constraint angle is 180 degrees - which in effect means that no constraint is applied. Ball joints only have a single constraint angle which describes the 'rotor' arc the joint allows, while hinges both have clockwise and anticlockwise constraint angles which may be enforced about a given reference axis that falls within the plane of the hinge.

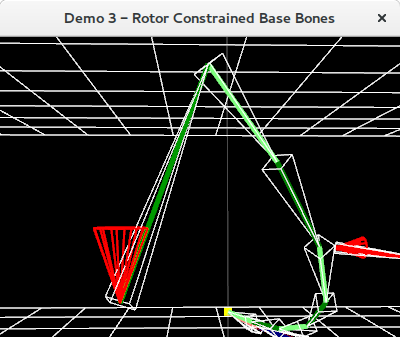
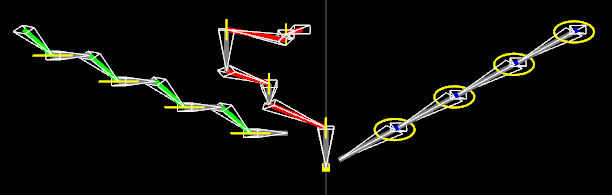
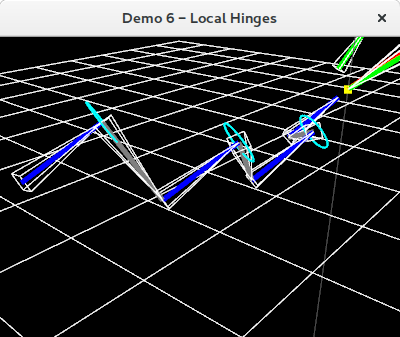


Figure 6 - Rotor / ball-joint constraints (top left) allow movement up to a given angle about the rotor constraint axis, local hinge constraints allow movement about the hinge axis in the coordinate system of the previous bone in the chain, while global hinge constraints (bottom) allow movement about a given world-space axis only.

### Solving IK Chains and Structures

Once a chain exists and contains one or more bones, it can be solved by calling the **solveForTarget()** method and providing it a target location to solve the chain for. Targets may be specified as Vec2f or Vec3f objects, or as floats.

If multiple chains exists in a structure, then calling **solveForTarget()** along with a target location results in each chain in the structure being solved in a first-in-first-solved manner for the same target location. However, if a chain in a structure has the **useEmbeddedTarget** mode enabled, then the chain will be solved for its own embedded target location rather than any provided target. In this manner, a structure may contain a combination of chains which can be solved for the specified target or their own embedded target location with a single call to **solveForTarget()** on the structure. If all chains in a structure use embedded targets then the provided target location is effectively ignored.

Chains that use embedded targets may update their targets via the chain’s **updateEmbededTarget()** method, and individual chains may be solved for their embedded targets via a call to **solveForEmbeddedTarget()**.

### Vectors and Matrices

The Caliko library uses a series of custom 2D and 3D vector classes (Vec2f, Vec3f) 3x3 and 4x4 matrices (Mat3f, Mat4f). These classes can be found in the **au.edu.federation.utils** package.

Array getters and setters exist to allow you to interact via the vector/matrix classes of your choice - but all Caliko vector and matrix properties are publicly accessible for ease of customisation purposes.

Both orthographic (2D) and perspective (3D) matrices can be created with ease through the provided Mat4f class, and the Vec3f and Vec2f classes offer a wide variety of methods to assist with manipulation of directions and locations. Please see the class source code and provided Javadoc for full API specifications.

### Reproducible Sequences

If you'd like to experiment with modifications to the Caliko source code, in particular, the results of solving 3D chains in a reproducible manner - then you can set a fixed seed for the random number generator (which is used by the MovingTarget3D class) by calling:

Utils.setSeed(SOME\_FIXED\_INTEGER);

For example:

Utils.setSeed(123);

### Obtaining Pitch, Yaw and Roll Angles for 3D Vectors / Bones

The Vec3f class has **getGlobalPitchDegs()** and **getGlobalYawDegs()** methods which provide the pitch and yaw, and the FabrikBone3f class has the same functions which just call those Vec3f methods on the direction of the bone.

Obtaining ***roll*** is not directly supported at the present time because 3D bones are stored as a two points in space, which is insufficient to calculate consistent roll angles. To reliably keep track of roll, the Caliko library must be refactored to store the 3D bone directions in quaternions, or it must store a rotation matrix along with each bone (as opposed to the current system of generating a valid but not guaranteed consistent rotation matrix on demand).

# Usage Examples

The Caliko demo application provides a number of usage examples which exercise various aspects of the library functionality. The examples below demonstrate how a variety of IK scenarios may be constructed.

In these 2D examples, UP/DOWN/LEFT/RIGHT are defined as the vectors (0.0f, 1.0f) / (0.0f, -1.0f) / (-1.0f, 0.0f) and (1.0f, 0.0f) respectively. In the 3D examples X\_AXIS is (1.0f, 0.0f, 0.0f), the Y\_AXIS is (0.0f, 1.0f, 0.0f) and Z\_AXIS is (0.0f, 0.0f, 1.0f). The Z-axis points directly outwards from the screen.

### 2D Demo 1 - Chain with GLOBAL\_ABSOLUTE basebone constraint and joint constraints

// Instantiate our structure, create a new chain and set it as fixed-base

FabrikStructure2D structure **=** **new** FabrikStructure2D**();**

FabrikChain2D chain **=** **new** FabrikChain2D**();**

chain**.**setFixedBaseMode**(true);**

// Default bone length

float boneLength **=** 40.0f**;**

// Create the basebone, set constraints and add it to the chain

FabrikBone2D base **=** **new** FabrikBone2D**(new** Vec2f**(**0.0f**,** -boneLength**),** **new** Vec2f**()** **);**

base**.**setClockwiseConstraintDegs**(**25.0f**);**

base**.**setAnticlockwiseConstraintDegs**(**90.0f**);**

chain**.**addBone**(**base**);**

// Fix the basebone and constrain it to the positive Y-axis

chain**.**setBaseboneConstraintType**(**BaseboneConstraintType2D**.**GLOBAL\_ABSOLUTE**);**

chain**.**setBaseboneConstraintUV**(**UP**);**

// Create and add the second bone - 50 clockwise, 90 anticlockwise

chain**.**addConsecutiveConstrainedBone**(**UP**,** boneLength**,** 50.0f**,** 90.0f**);**

// Create and add the third bone - 75 clockwise, 90 anticlockwise

chain**.**addConsecutiveConstrainedBone**(**UP**,** boneLength**,** 75.0f**,** 90.0f**);**

// Finally, add the chain to the structure

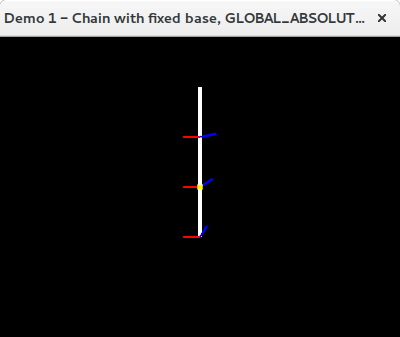
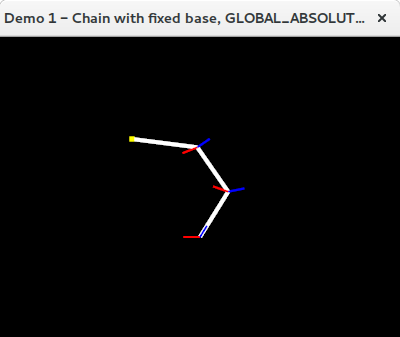
structure**.**addChain**(**chain**);**

Figure 7 - Chain with GLOBAL\_ABSOLUTE basebone constraint and joint constraints (left) original configuration (right) example solve configuration

### 2D Demo 2 - Chain with fixed base and multiple unconstrained bones

// Instantiate our structure, create a new chain and set it as fixed-base

FabrikStructure2D structure **=** **new** FabrikStructure2D**();**

FabrikChain2D chain **=** **new** FabrikChain2D**();**

chain**.**setFixedBaseMode**(true);**

// Create the basebone and add it to the chain.

// Params: Start location, direction, length

FabrikBone2D base **=** **new** FabrikBone2D**(new** Vec2f**(),** RIGHT**,** 10.0f**);**

chain**.**addBone**(**base**);**

// Add a series of additional bones which initially form a circle

float boneLength **=** 10.0f**;**

float numBones **=** 15**;**

float rotStep **=** 360.0f **/** numBones**;**

**for** **(**int loop **=** 0**;** loop **<** numBones**;** loop**++){**

// Rotate each bone by 10 degrees and add it to the chain

Vec2f rotatedUV **=** Vec2f**.**rotateDegs**(**RIGHT**,** loop **\*** rotStep**);**

chain**.**addConsecutiveBone**(**rotatedUV**,** boneLength**);**

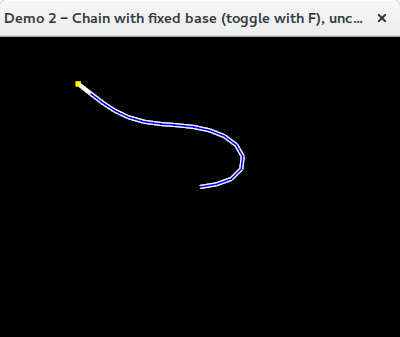
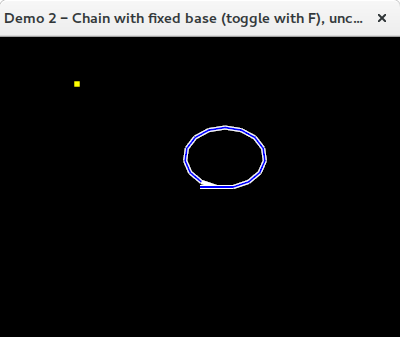
**}**

// Finally, add the chain to the structure

structure**.**addChain**(**chain**);**

### 2D Demo 3 - Chain with fixed base and multiple constrained bones

Figure 8 - Unconstrained chain (left) original configuration (right) example solve configuration



// Instantiate our structure, create a new chain and set it as fixed-base

FabrikStructure2D structure **=** **new** FabrikStructure2D**();**

FabrikChain2D chain **=** **new** FabrikChain2D**();**

chain**.**setFixedBaseMode**(true);**

// Create the basebone and add it to the chain.

// Params: Start location, direction, length

FabrikBone2D base **=** **new** FabrikBone2D**(new** Vec2f**(),** RIGHT**,** 10.0f**);**

chain**.**addBone**(**base**);**

// Add a series of additional bones which initially form a circle

float boneLength **=** 10.0f**;**

float numBones **=** 15**;**

float rotStep **=** 360.0f **/** numBones**;**

**for** **(**int loop **=** 0**;** loop **<** numBones**;** **++**loop**) {**

// Rotate each bone by 10 degrees and add it to the chain with constraint angles

// Note: Anticlockwise rotation is positive, clockwise rotation is negative.

Vec2f rotatedUV **=** Vec2f**.**rotateDegs**(**RIGHT**,** loop **\*** rotStep**);**

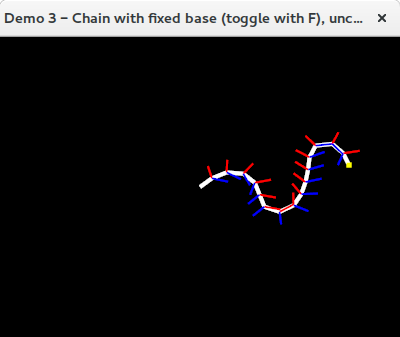
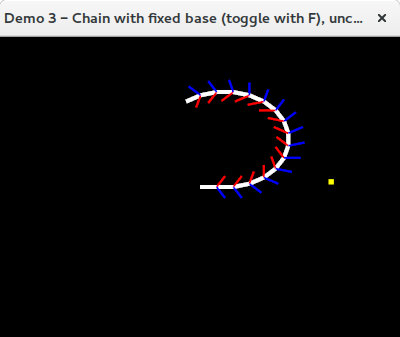
chain**.**addConsecutiveConstrainedBone**(**rotatedUV**,** boneLength**,** 60.0f**,** 60.0f**);**

**}**

// Finally, add the chain to the structure

structure**.**addChain**(**chain**);**

Figure 9 - Constrained chain in (left) original configuration and (right) example solve configuration. No bone may exceed its anticlockwise or clockwise constraint angles, shown in red and blue respectively.



### 2D Demo 4 - Chain with fixed base and multiple constrained bones

// Instantiate our FabrikStructure2D

FabrikStructure2D structure **=** **new** FabrikStructure2D**();**

// ----- Vertical chain -----

float boneLength **=** 50.0f**;**

FabrikChain2D verticalChain **=** **new** FabrikChain2D**();**

FabrikBone2D basebone **=** **new** FabrikBone2D**(** **new** Vec2f**(**0.0f**,** **-**50.0f**),** UP**,** boneLength**);**

// Note: Default basebone constraint type is NONE

verticalChain**.**addBone**(**basebone**);**

// Add two additional consecutive bones

verticalChain**.**addConsecutiveConstrainedBone**(**UP**,** boneLength**,** 90.0f**,** 90.0f**);**

verticalChain**.**addConsecutiveConstrainedBone**(**UP**,** boneLength**,** 90.0f**,** 90.0f**);**

// Add our main chain to structure

structure**.**addChain**(**verticalChain**);**

// ----- Left branch chain -----

boneLength **=** 30.0f**;**

// Create the base bone and set its colour

basebone **=** **new** FabrikBone2D**(** **new** Vec2f**(),** **new** Vec2f**(-**boneLength**,** 0.0f**)** **);**

basebone**.**setColour**(**Utils**.**MID\_GREEN**);**

// Create the chain and add the basebone to it

FabrikChain2D leftChain **=** **new** FabrikChain2D**();**

leftChain**.**addBone**(**basebone**);**

// Add consecutive constrained bones

// Note: The base-bone is unconstrained, but these bones ARE constrained

leftChain**.**addConsecutiveConstrainedBone**(**LEFT**,** boneLength**,** 90.0f**,** 90.0f**,** Utils**.**MID\_GREEN**);**

leftChain**.**addConsecutiveConstrainedBone**(**LEFT**,** boneLength**,** 90.0f**,** 90.0f**,** Utils**.**MID\_GREEN**);**

// Add the chain to the structure, connecting to the end of bone 0 in chain 0

structure**.**addConnectedChain**(**leftChain**,** 0**,** 0**,** BoneConnectionPoint2D**.**END**);**

// ----- Right branch chain -----

// Create the base bone

basebone **=** **new** FabrikBone2D**(** **new** Vec2f**(),** **new** Vec2f**(**boneLength**,** 0.0f**)** **);**

basebone**.**setColour**(**Utils**.**GREY**);**

// Create the chain and add the basebone to it

FabrikChain2D rightChain **=** **new** FabrikChain2D**();**

rightChain**.**addBone**(**basebone**);**

// Add two consecutive constrained bones to the chain

// Note: The base-bone is unconstrained, but these bones ARE constrained

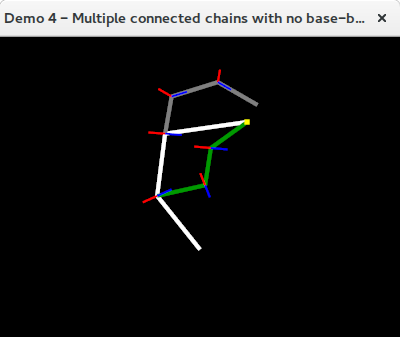
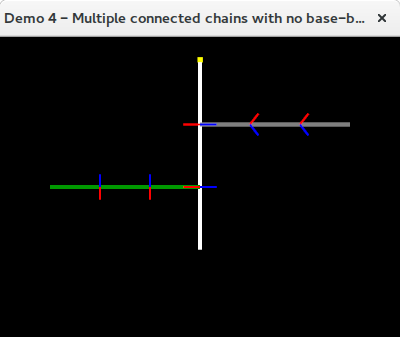
rightChain**.**addConsecutiveConstrainedBone**(**RIGHT**,** boneLength**,** 60.0f**,** 60.0f**,** Utils**.**GREY**);**

rightChain**.**addConsecutiveConstrainedBone**(**RIGHT**,** boneLength**,** 60.0f**,** 60.0f**,** Utils**.**GREY**);**

// Add the chain to the structure, connecting to the end of bone 1 in chain 0

structure**.**addConnectedChain**(**rightChain**,** 0**,** 2**,** BoneConnectionPoint2D**.**START**);**

Figure 10 - Connected chains with fixed basebones, joint constraints and no basebone constraints. (left) original configuration (right) example solve configuration. The joint constraints on the grey chain prevent a successful solve in this instance.



### 2D Demo 5 - Multiple connected chains with LOCAL\_RELATIVE basebone constraints

FabrikStructure2D structure **=** **new** FabrikStructure2D**();**

// ----- Vertical chain -----

float boneLength **=** 50.0f**;**

FabrikChain2D verticalChain **=** **new** FabrikChain2D**();**

FabrikBone2D basebone **=** **new** FabrikBone2D**(** **new** Vec2f**(**0.0f**,** **-**100.0f**),** UP**,** boneLength**);**

basebone**.**setClockwiseConstraintDegs**(**15.0f**);**

basebone**.**setAnticlockwiseConstraintDegs**(**15.0f**);**

verticalChain**.**addBone**(**basebone**);**

verticalChain**.**addConsecutiveConstrainedBone**(**UP**,** boneLength**,** 30.0f**,** 30.0f**);**

verticalChain**.**addConsecutiveConstrainedBone**(**UP**,** boneLength**,** 30.0f**,** 30.0f**);**

verticalChain**.**setBaseboneConstraintType**(**BaseboneConstraintType2D**.**GLOBAL\_ABSOLUTE**);**

verticalChain**.**setBaseboneConstraintUV**(**UP**);**

// Add chain to structure

structure**.**addChain**(**verticalChain**);**

// ----- Left branch chain -----

boneLength **=** 30.0f**;**

// Create the base bone

basebone **=** **new** FabrikBone2D**(** **new** Vec2f**(),** **new** Vec2f**(-**boneLength**,** 0.0f**)** **);**

basebone**.**setClockwiseConstraintDegs**(**60.0f**);**

basebone**.**setAnticlockwiseConstraintDegs**(**60.0f**);**

basebone**.**setColour**(**Utils**.**MID\_GREEN**);**

// Create the chain, add the basebone and enable base bone constraint mode

FabrikChain2D leftChain **=** **new** FabrikChain2D**();**

leftChain**.**setBaseboneConstraintType**(**BaseboneConstraintType2D**.**LOCAL\_RELATIVE**);**

leftChain**.**addBone**(**basebone**);**

// Add consecutive constrained bones

leftChain**.**addConsecutiveConstrainedBone**(**LEFT**,** boneLength**,** 60.0f**,** 60.0f**,**  Utils**.**MID\_GREEN**);**

leftChain**.**addConsecutiveConstrainedBone**(**LEFT**,** boneLength**,** 60.0f**,** 60.0f**,**  Utils**.**MID\_GREEN**);**

// Add the chain to the structure, connecting at the end of bone 0 in chain 0

structure**.**addConnectedChain**(**leftChain**,** 0**,** 0**,** BoneConnectionPoint2D**.**END**);**

// ----- Right branch chain -----

// Create the base bone

basebone **=** **new** FabrikBone2D**(** **new** Vec2f**(),** **new** Vec2f**(**boneLength**,** 0.0f**)** **);**

basebone**.**setClockwiseConstraintDegs**(**30.0f**);**

basebone**.**setAnticlockwiseConstraintDegs**(**30.0f**);**

basebone**.**setColour**(**Utils**.**GREY**);**

// Create the chain, add the basebone and enable base bone constraint mode

FabrikChain2D rightChain **=** **new** FabrikChain2D**();**

rightChain**.**setBaseboneConstraintType**(**BaseboneConstraintType2D**.**LOCAL\_RELATIVE**);**

rightChain**.**addBone**(**basebone**);**

// Add two consecutive constrained bones to the chain

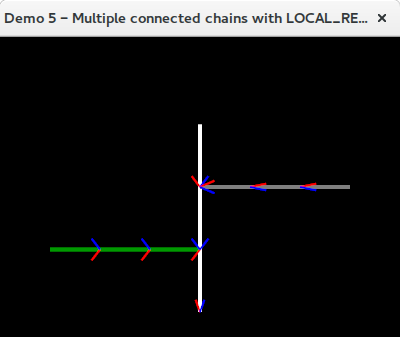
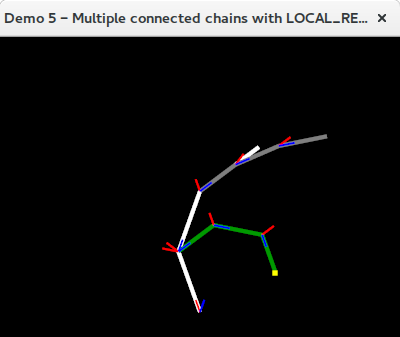
rightChain**.**addConsecutiveConstrainedBone**(**RIGHT**,** boneLength**,** 15.0f**,** 15.0f**,** Utils**.**GREY**);**

rightChain**.**addConsecutiveConstrainedBone**(**RIGHT**,** boneLength**,** 15.0f**,** 15.0f**,** Utils**.**GREY**);**

// Add the chain to the structure, connecting at the end of bone 1 in chain 0

structure**.**addConnectedChain**(**rightChain**,** 0**,** 1**,** BoneConnectionPoint2D**.**END**);**

Figure 11 - Connected chains with local relative basebone constraints. (left) original configuration (right) example solved configuration. The white and grey chains are highly constrained in this example. The bottom row shows how the basebone constraints of the green chain move relative to the direction of the first bone in the white chain to which it is attached. Notice that the relative basebone constraints in the original configuration are not being honoured - this is because the structure as a whole has not been solved yet. When a solve attempt is made, all constraints are honoured.



### 2D Demo 6 - Multiple connected chains with LOCAL\_ABSOLUTE basebone constraints

While LOCAL\_RELATIVE basebone constraints are applied relative to the direction of the bone a chain is connected to, LOCAL\_ABSOLUTE basebone constrains are applied as directions in the local coordinate space of the bone a chain is connected to.

For example, if a basebone constraint is the vector (-1.0f, 0.0f) [i.e. LEFT], then:

* If the bone the chain is connected to is pointing directly upwards, then the 'local' left remains ***left***,
* If the bone the chain is connected to is pointing directly left, then 'local' left is now ***down***,
* If the bone the chain is connected to is pointing directly right, then 'local' left is now ***up***.

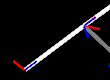
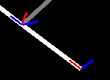
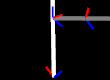


Figure 12 - The LOCAL\_ABSOLUTE basebone constraint on the grey chain is set to RIGHT, but is maintained in the local space of the white 'host' bone that it's connected to.

LOCAL\_ABSOLUTE basebone constraints can be set on a chain by specifying the constraint type **and** a constraint direction unit vector:

FabrikChain2D chain **=** **new** FabrikChain2D**();**

chain**.**setBaseboneConstraintType**(**BaseboneConstraintType2D**.**LOCAL\_ABSOLUTE**);**

chain**.**setBaseboneConstraintUV**(**RIGHT**);**

// ...add bones to the chain here...

// If a chain has LOCAL basebone constraints it MUST be connected to another chain!

structure**.**addConnectedChain**(c**hain**,** 0**,** 0**,** BoneConnectionPoint2D**.**END**);**

### 2D Demo 7 – Chains with embedded targets

float boneLength **=** 50.0f**;**

float startY **=** **-**100.0f**;**

mStructure **=** **new** FabrikStructure2D**();**

FabrikChain2D chain **=** **new** FabrikChain2D**();**

// ----- Central white chain ------

// Create the first bone, configure it, and add it to the chain

FabrikBone2D basebone**;**

basebone **=** **new** FabrikBone2D**(new** Vec2f**(**0.0f**,** startY**),** **new** Vec2f**(**0.0f**,** startY **+** boneLength**)** **);**

basebone**.**setClockwiseConstraintDegs**(**65.0f**);**

basebone**.**setAnticlockwiseConstraintDegs**(**65.0f**);**

chain**.**addBone**(**basebone**);**

// Fix basebone to its current location, and constrain it to the positive Y-axis

chain**.**setFixedBaseMode**(true);**

chain**.**setBaseboneConstraintType**(**BaseboneConstraintType2D**.**GLOBAL\_ABSOLUTE**);**

chain**.**setBaseboneConstraintUV**(**UP**);**

// Add second and third bones

chain**.**addConsecutiveBone**(**UP**,** boneLength**);**

chain**.**addConsecutiveBone**(**UP**,** boneLength**);**

// Finally, add the chain to the structure

mStructure**.**addChain**(**chain**);**

// ----- Left green chain with embedded target -----

FabrikChain2D leftChain **=** **new** FabrikChain2D**();**

leftChain**.**setEmbeddedTargetMode**(true);** // Embedded target loc. set in demo loop

basebone **=** **new** FabrikBone2D**(new** Vec2f**(),** **new** Vec2f**(-**boneLength **/** 6.0f**,** 0.0f**)** **);**

// Add fifteen bones

leftChain**.**addBone**(**basebone**);**

**for** **(**int boneLoop **=** 0**;** boneLoop **<** 15**;** **++**boneLoop**)**

**{**

leftChain**.**addConsecutiveConstrainedBone**(**RIGHT**,** boneLength **/** 6.0f**,** 25.0f**,** 25.0f**);**

**}**

// Set chain colour and basebone constraint type

leftChain**.**setColour**(**Utils**.**MID\_GREEN**);**

// Add the left chain to the structure, connected to the start of bone 1 in chain 0

mStructure**.**addConnectedChain**(**leftChain**,** 0**,** 1**,** BoneConnectionPoint2D**.**START**);**

// ----- Right grey chain with embedded target -----

FabrikChain2D rightChain **=** **new** FabrikChain2D**();**

rightChain**.**setEmbeddedTargetMode**(true);** // Embedded target loc. set in demo loop basebone **=** **new** FabrikBone2D**(new** Vec2f**(),** **new** Vec2f**(**boneLength **/** 5.0f**,** 0.0f**)** **);**

basebone**.**setClockwiseConstraintDegs**(**60.0f**);**

basebone**.**setAnticlockwiseConstraintDegs**(**60.0f**);**

// Add ten bones

rightChain**.**addBone**(**basebone**);**

**for** **(**int boneLoop **=** 0**;** boneLoop **<** 10**;** **++**boneLoop**)**

**{**

rightChain**.**addConsecutiveBone**(**RIGHT**,** boneLength **/** 5.0f**);**

**}**

// Set chain colour and basebone constraint type

rightChain**.**setColour**(**Utils**.**GREY**);**

rightChain**.**setBaseboneConstraintType**(**BaseboneConstraintType2D**.**LOCAL\_ABSOLUTE**);**

rightChain**.**setBaseboneRelativeConstraintUV**(**RIGHT**);**

// Add the right chain to the structure, connected to the start of bone 2 in chain  
// 0

mStructure**.**addConnectedChain**(**rightChain**,** 0**,** 2**,** BoneConnectionPoint2D**.**START**);**

Chains that use embedded targets can have those targets updated by calls to the **updateEmbeddedTarget()** method.

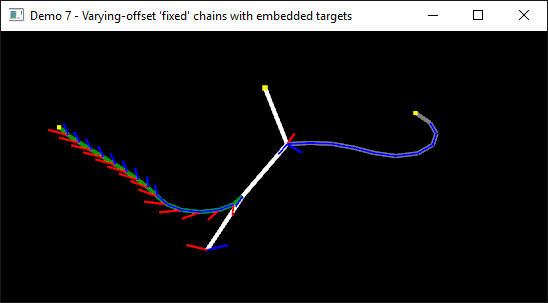


Figure 13 - Multiple connected chains with 'fixed' bases. The left and right chains use embedded targets which are updated separately via calls to the chain's updateEmbeddedTarget method.

### 2D Demo 8 – Multiple Nested Chains in a Semi Random Configuration

Just an example of nesting chains.

// Instantiate our FabrikStructure2D

**this.**structure **=** **new** FabrikStructure2D**(**"Demo 8 - Multiple nested chains in a semi-random configuration"**);**

**this.**structure**.**addChain**(** createRandomChain**()** **);**

int chainsInStructure **=** 1**;**

int maxChains **=** 3**;**

**for** **(**int chainLoop **=** 0**;** chainLoop **<** maxChains**;** chainLoop**++)**

**{**

FabrikChain2D tempChain **=** createRandomChain**();**

tempChain**.**setBaseboneConstraintType**(**BaseboneConstraintType2D**.**LOCAL\_RELATIVE**);**

tempChain**.**setBaseboneConstraintUV**(**UP**);**

**this.**structure**.**connectChain**(** createRandomChain**(),** Utils**.**randRange**(**0**,** chainsInStructure**++),** Utils**.**randRange**(**0**,** 5**)** **);**

**}**

private FabrikChain2D createRandomChain**()**

**{**

float boneLength **=** 20.0f**;**

float boneLengthRatio **=** 0.8f**;**

float constraintAngleDegs **=** 20.0f**;**

float constraintAngleRatio **=** 1.4f**;**

// ----- Vertical chain -----

FabrikChain2D chain **=** **new** FabrikChain2D**();**

chain**.**setFixedBaseMode**(true);**

FabrikBone2D basebone **=** **new** FabrikBone2D**(** **new** Vec2f**(),** UP**,** boneLength**);**

basebone**.**setClockwiseConstraintDegs**(**constraintAngleDegs**);**

basebone**.**setAnticlockwiseConstraintDegs**(**constraintAngleDegs**);**

chain**.**setBaseboneConstraintType**(**BaseboneConstraintType2D**.**LOCAL\_RELATIVE**);**

chain**.**addBone**(**basebone**);**

int numBones **=** 6**;**

float perturbLimit **=** 0.4f**;**

**for** **(**int boneLoop **=** 0**;** boneLoop **<** numBones**;** boneLoop**++)**

**{**

boneLength **\*=** boneLengthRatio**;**

constraintAngleDegs **\*=** constraintAngleRatio**;**

Vec2f perturbVector **=** **new** Vec2f**(** Utils**.**randRange**(-**perturbLimit**,** perturbLimit**),** Utils**.**randRange**(-**perturbLimit**,** perturbLimit**)** **);**

chain**.**addConsecutiveConstrainedBone**(** UP**.**plus**(**perturbVector**),** boneLength**,** constraintAngleDegs**,** constraintAngleDegs **);**

**}**

chain**.**setColour**(** Colour4f**.**randomOpaqueColour**()** **);**

**return** chain**;**

**}**

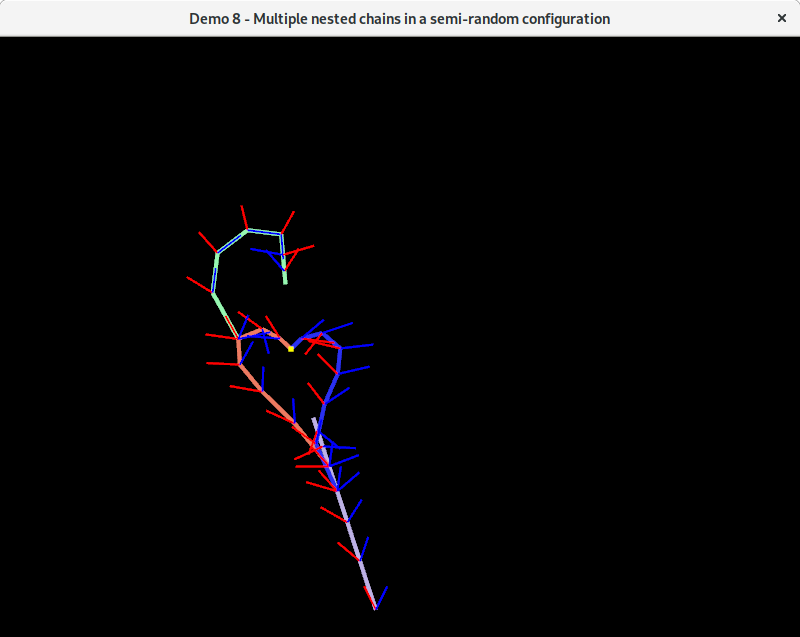


Figure 14 - Multiple nested chains all being solved for the same target location.

### 2D Demo 9 - World-Space 2D Constraints

An example of specifying that 2D joint constraints should be interpreted in world-space coordinates rather than in the coordinate system of the previous bone in the chain (which is the default).

// Instantiate our FabrikStructure2D

**this.**structure **=** **new** FabrikStructure2D**(**"Demo 9 - Chain with fixed base and world space (GLOBAL) bone constaints."**);**

// Create a new chain

FabrikChain2D chain **=** **new** FabrikChain2D**();**

float boneLength **=** 40.0f**;**

// Create and add first bone - 25 clockwise, 90 anti-clockwise

FabrikBone2D basebone**;**

basebone **=** **new** FabrikBone2D**(new** Vec2f**(**0.0f**,** **-**boneLength**),** **new** Vec2f**(**0.0f**,** 0.0f**)** **);**

basebone**.**setClockwiseConstraintDegs**(**90.0f**);**

basebone**.**setAnticlockwiseConstraintDegs**(**90.0f**);**

chain**.**addBone**(**basebone**);**

// Fix the base bone to its current location, and constrain it to the positive Y-axis

chain**.**setFixedBaseMode**(true);**

chain**.**setBaseboneConstraintType**(**BaseboneConstraintType2D**.**GLOBAL\_ABSOLUTE**);**

chain**.**setBaseboneConstraintUV**(** **new** Vec2f**(**0.0f**,** 1.0f**)** **);**

chain**.**addConsecutiveBone**(**UP**,** boneLength**);**

chain**.**addConsecutiveBone**(**UP**,** boneLength**);**

// Create and add the fourth 'gripper' bone - locked in place facing right (i.e. 0  
// degree movement allowed both clockwise & anti-clockwise)

// Note: The start location of (50.0f, 50.0f) is ignored because we're going to add  
// this to the end of the chain, wherever that may be.

FabrikBone2D gripper **=** **new** FabrikBone2D**(new** Vec2f**(**50.0f**,** 50.0f**),** RIGHT**,** boneLength **/** 2.0f**,** 30.0f**,** 30.0f**);**

gripper**.**setJointConstraintCoordinateSystem**(**ConstraintCoordinateSystem**.**GLOBAL**);**

gripper**.**setGlobalConstraintUV**(**RIGHT**);**

chain**.**addConsecutiveBone**(**gripper**);**

// Finally, add the chain to the structure

**this.**structure**.**addChain**(**chain**);**

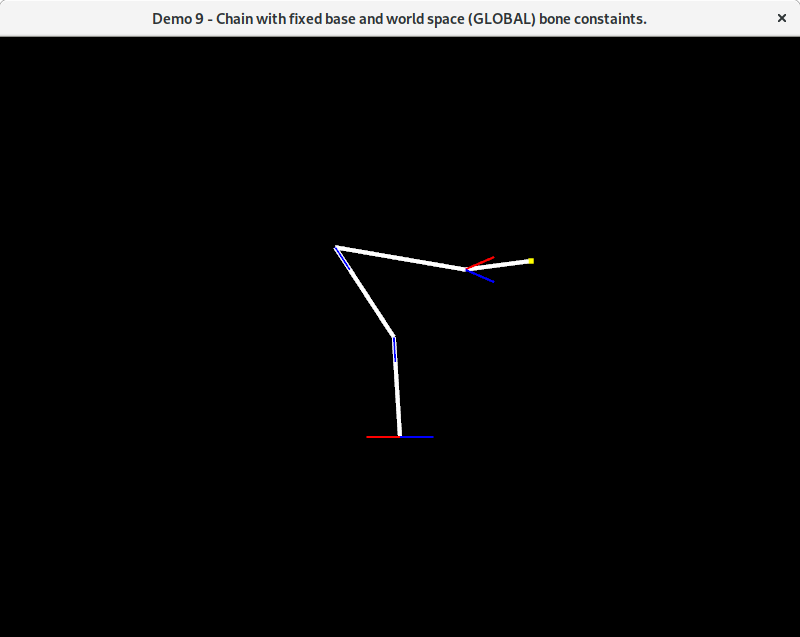


Figure 15 - An example of creating a chain with bones where the constraint angles are always treated to be about a world-space direction rather than relative to the direction of the previous bone in the chain.

### 3D Demo 1 - Unconstrained bones

FabrikStructure3D structure **=** **new** FabrikStructure3D**();**

FabrikChain3D chain **=** **new** FabrikChain3D**();**

Colour4f colour **=** **new** Colour4f**(**Utils**.**GREEN**);**

float boneLength **=** 10.0f**;**

Vec3f boneDirection **=** Z\_AXIS.negated**();** // i.e. into the screen

// Create a basebone and then add it to the chain

Vec3f start **=** **new** Vec3f**(**0.0f**,** 0.0f**,** 40.0f**);**

Vec3f end **=** start**.**plus**(** defaultBoneDirection**.**times**(**boneLength**)** **);**

FabrikBone3D basebone **=** **new** FabrikBone3D**(**start**,** end**);**

basebone**.**setColour**(**colour**);**

chain**.**addBone**(**basebone**);**

// Add additional consecutive, unconstrained bones to the chain

**for** **(**int boneLoop **=** 0**;** boneLoop **<** 7**;** boneLoop**++) {**

colour **=** **(**boneLoop **%** 2 **==** 0**)** **?** colour**.**lighten**(**0.4f**)** **:** colour**.**darken**(**0.4f**);**

chain**.**addConsecutiveBone**(**boneDirection**,** boneLength**,** colour**);**

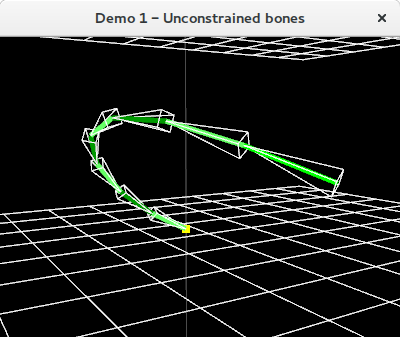
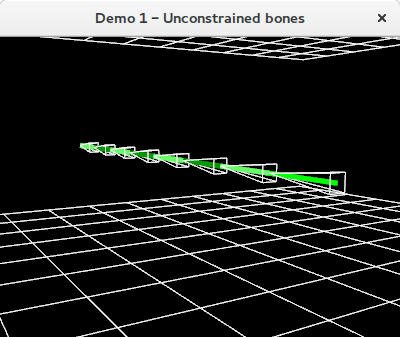
**}**

// Finally, add the chain to the structure

structure**.**addChain**(**chain**);**

3D Demo 2 - Rotor / ball joint constrained bones

Figure 16 - A 3D chain containing unconstrained bones (left) in their initial and (right) in an example solved state.



FabrikStructure3D structure **=** **new** FabrikStructure3D**();**

int numChains **=** 3**;**

float rotStep **=** 360.0f **/** **(**float**)**numChains**;**

float constraintDegs **=** 45.0f**;**

Colour4f colour **=** **new** Colour4f**();**

float boneLength **=** 10.0f**;**

Vec3f boneDirection **=** Z\_AXIS.negated**();** // i.e. into the screen

// Create chains and set colours

**for** **(**int chainLoop **=** 0**;** chainLoop **<** numChains**;** **++**chainLoop**)** **{**

FabrikChain3D chain **=** **new** FabrikChain3D**();**

**switch** **(**chainLoop **%** numChains**)** **{**

**case** 0**:** boneColour**.**set**(**Utils**.**MID\_RED**);** **break;**

**case** 1**:** boneColour**.**set**(**Utils**.**MID\_GREEN**);** **break;**

**case** 2**:** boneColour**.**set**(**Utils**.**MID\_BLUE**);** **break;**

**}**

// Set up the initial base bone location...

Vec3f startLoc **=** **new** Vec3f**(**0.0f**,** 0.0f**,** **-**40.0f**);**

startLoc **=** Vec3f**.**rotateYDegs**(**startLoc**,** rotStep **\*** **(**float**)**chainLoop**);**

Vec3f endLoc **=** **new** Vec3f**(**startLoc**);**

endLoc**.**z **-=** defaultBoneLength**;**

// ...then create a base bone, set its colour and add it to the chain.

FabrikBone3D basebone **=** **new** FabrikBone3D**(**startLoc**,** endLoc**);**

basebone**.**setColour**(**colour**);**

chain**.**addBone**(**basebone**);**

// Add additional consecutive rotor constrained bones to the chain **for** **(**int boneLoop **=** 0**;** boneLoop **<** 7**;** **++**boneLoop**) {**

colour **=** **(**boneLoop **%** 2 **==** 0**)** **?** colour**.**lighten**(**0.4f**)** **:**  colour**.**darken**(**0.4f**);** chain**.**addConsecutiveRotorConstrainedBone**(**boneDirection**,** boneLength**,**  constraintDegs**,** colour**);**

**}**

// Finally, add the chain to the structure

structure**.**addChain**(**chain**);**

**}**

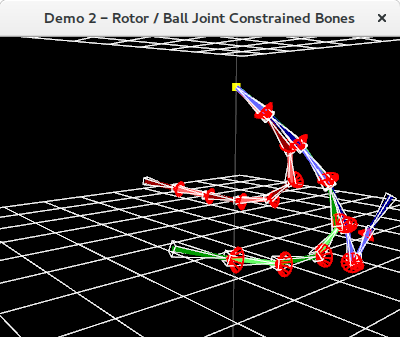
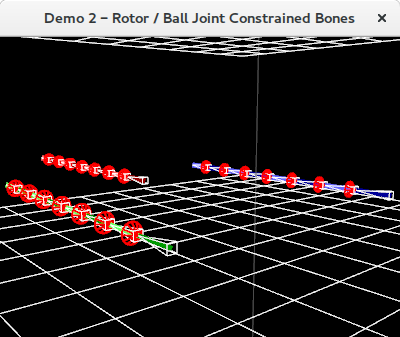


Figure 17 - Chains with 45 degree rotor constrained bones (left) in their initial state and (right) in an example solved state.

### 3D Demo 3 - Chains with rotor constrained basebones

FabrikStructure3D structure **=** **new** FabrikStructure3D**();**

int numChains **=** 3**;**

float rotStep **=** 360.0f **/** **(**float**)**numChains**;**

Colour4f colour **=** **new** Colour4f**();**

Colour4f bbColour **=** **new** Colour4f**();**

Vec3f bbConstraintAxis **=** **new** Vec3f**();**

float bbConstraintDegs **=** 20.0f**;**

**for** **(**int chainLoop **=** 0**;** chainLoop **<** numChains**;** **++**chainLoop**)** **{** // Set bone colours and basebone constraint axes

**switch** **(**chainLoop **%** 3**) {**

**case** 0**:**

colour**.**set**(**Utils**.**MID\_RED**);**

bbColour**.**set**(**Utils**.**RED**);**

bbConstraintAxis **=** X\_AXIS**;**

**break;**

**case** 1**:**

colour**.**set**(**Utils**.**MID\_GREEN**);**

bbColour**.**set**(**Utils**.**MID\_GREEN**);**

bbConstraintAxis **=** Y\_AXIS**;**

**break;**

**case** 2**:**

colour**.**set**(**Utils**.**MID\_BLUE**);**

bbColour**.**set**(**Utils**.**BLUE**);**

bbConstraintAxis **=** Z\_AXIS**.**negated**();**

**break;**

**}**

// Create a new chain and set up the basebone start/end locations

FabrikChain3D chain **=** **new** FabrikChain3D**();**

Vec3f start **=** **new** Vec3f**(**0.0f**,** 0.0f**,** **-**40.0f**);**

start **=** Vec3f**.**rotateYDegs**(**start**,** rotStep **\*** **(**float**)**chainLoop**);**

Vec3f end **=** start**.**plus**(** bbConstraintAxis**.**times**(**defaultBoneLength **\*** 2.0f**) );**

// ...then create a base bone, set its colour, add it to the chain FabrikBone3D basebone **=** **new** FabrikBone3D**(**start**,** end**);**

basebone**.**setColour**(**baseBoneColour**);**

chain**.**addBone**(**basebone**);**

// EITHER: Set the basebone to be global rotor constrained: chain**.**setRotorBaseboneConstraint**(**BaseboneConstraintType3D**.**GLOBAL\_ROTOR**,**  baseBoneConstraintAxis**,** baseBoneConstraintAngleDegs**);**

// OR: Freely-rotating global hinge constrained:

chain**.**setFreelyRotatingGlobalHingedBasebone**(**Y\_AXIS**);**

// OR: Non-freely-rotating global hinge constrained

// Params: hinge axis, clockwise angle, anticlockwise angle, reference axis

chain**.**setGlobalHingeBaseboneConstraint**(**Y\_AXIS**,** 90.0f**,** 45.0f**,** X\_AXIS**);**

// Add additional consecutive, unconstrained bones to the chain

**for** **(**int boneLoop **=** 0**;** boneLoop **<** 7**;** **++**boneLoop**) {**

colour **=** **(**boneLoop **%** 2 **==** 0**)** **?** colour**.**lighten**(**0.5f**)** **:**   
 colour**.**darken**(**0.5f**);**

chain**.**addConsecutiveBone**(**defaultBoneDirection**,** defaultBoneLength**,**  colour**);**

**}**

// Finally, add the chain to the structure

structure**.**addChain**(**chain**);**

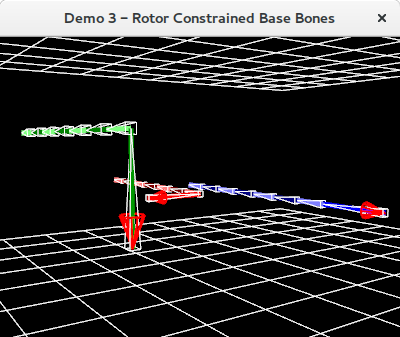
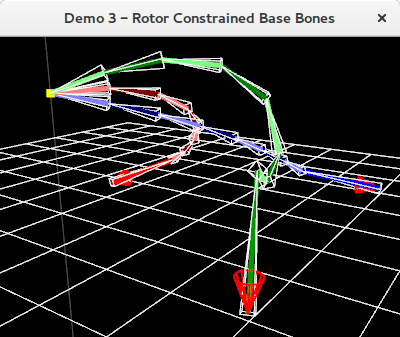
**}**

Figure 18 - Chains with rotor constrained basebones (left) in their initial configuration and (right) in an example solved configuration.

### 3D Demo 4 - Chains with freely rotating global hinges

FabrikStructure3D structure **=** **new** FabrikStructure3D**();**

int numChains **=** 3**;**

float rotStep **=** 360.0f **/** **(**float**)**numChains**;**

Vec3f boneDirection **=** Z\_AXIS.negated**();** // i.e. into the screen

float boneLength **=** 10.0f**;**

// Create a circular arrangement of 3 chains with varying global hinged bones

Vec3f hingeAxis **=** **new** Vec3f**();**

**for** **(**int chainLoop **=** 0**;** chainLoop **<** numChains**;** **++**chainLoop**) {**

// Set colour and axes

Colour4f chainColour **=** **new** Colour4f**();**

**switch** **(**chainLoop **%** numChains**) {**

**case** 0**:**

chainColour**.**set**(**Utils**.**RED**);** hingeAxis **=** X\_AXIS**; break;**

**case** 1**:**

chainColour**.**set**(**Utils**.**GREEN**);** hingeAxis **=** Y\_AXIS**; break;**

**case** 2**:**

chainColour**.**set**(**Utils**.**BLUE**);** hingeAxis **=** Z\_AXIS**; break;**

**}**

// Create a new chain and set the start and end locations...

FabrikChain3D chain **=** **new** FabrikChain3D**();**

Vec3f start **=** **new** Vec3f**(**0.0f**,** 0.0f**,** **-**40.0f**);**

start **=** Vec3f**.**rotateYDegs**(**start**,** rotStep **\*** **(**float**)**chainLoop**);**

Vec3f end **=** startLoc**.**plus**(** boneDirection**.**times**(**boneLength**)** **);**

// ...then create a base bone, set its colour, and add it to the chain.

FabrikBone3D basebone **=** **new** FabrikBone3D**(**startLoc**,** endLoc**);**

basebone**.**setColour**(**chainColour**);**

chain**.**addBone**(**basebone**);**

// Add alternating global hinge and unconstrained bones to the chain

**for** **(**int boneLoop **=** 0**;** boneLoop **<** 7**;** **++**boneLoop**) {**

**if** **(**boneLoop **%** 2 **==** 0**) {** chain**.**addConsecutiveFreelyRotatingHingedBone**(**boneDirection**,**  boneLength**,** JointType**.**GLOBAL\_HINGE**,** globalHingeAxis**,** Utils**.**GREY**);**

**} else** **{**

chain**.**addConsecutiveBone**(**boneDirection**,** boneLength**,** chainColour**);**

**}**

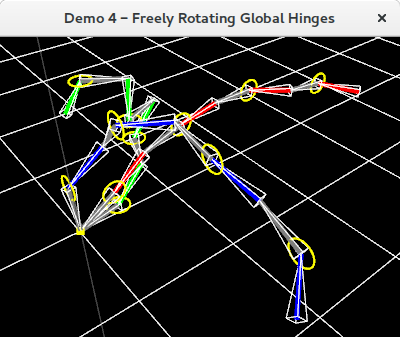
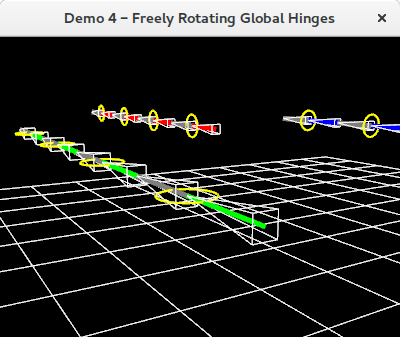
**}**

// Finally, add the chain to the structure

structure**.**addChain**(**chain**);**

**}**

Figure 19 - Chains with global hinged bones (left) in their original and (right) in an example solved configuration.



### 3D Demo 5 - Global hinges with reference axis constraints

FabrikStructure3D structure **=** **new** FabrikStructure3D**();**

FabrikChain3D chain **=** **new** FabrikChain3D**();**

Vec3d down = Y\_Axis.negated**();**

float boneLength **=** 10.0f**;**

// Set up the initial base bone location...

Vec3f startLoc **=** **new** Vec3f**(**0.0f**,** 30f**,** **-**40.0f**);**

Vec3f endLoc **=** **new** Vec3f**(**startLoc**);**

endLoc**.**y **-=** defaultBoneLength**;**

// ...then create a base bone, set its colour, and add it to the chain.

FabrikBone3D basebone **=** **new** FabrikBone3D**(**startLoc**,** endLoc**);**

basebone**.**setColour**(**Utils**.**YELLOW**);**

chain**.**addBone**(**basebone**);**

// Add alternating global hinge constrained, and unconstrained bones to the chain

float cwDegs **=** 120.0f**;**

float acwDegs **=** 120.0f**;**

**for** **(**int boneLoop **=** 0**;** boneLoop **<** 8**;** **++**boneLoop**) {**

**if** **(**boneLoop **%** 2 **==** 0**) {**

// Params: bone direction, bone length, joint type, hinge rotation  
 // axis, clockwise constraint angle, anticlockwise constraint angle,  
 // hinge constraint reference axis, colour

// Note: There is a version of this method where you do not specify the  
 // colour - the default is to draw the bone in white.

chain**.**addConsecutiveHingedBone**(**down**,** boneLength**,** JointType**.**GLOBAL\_HINGE,Z\_AXIS**,** cwDegs**,**acwDegs**,** down**,** Utils**.**GREY**);**

**} else {**

chain**.**addConsecutiveBonedown**(**down**,** defaultBoneLength**,** Utils**.**MID\_GREEN**);**

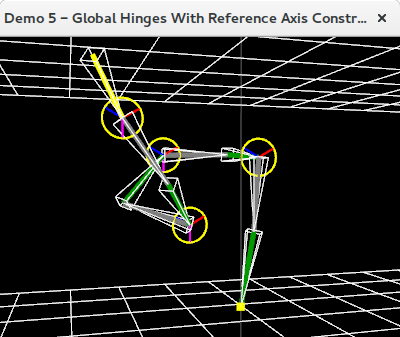
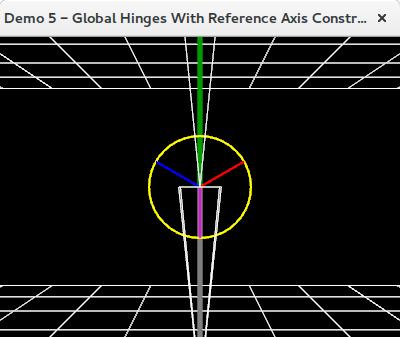
**}**

**}**

// Finally, add the chain to the structure

structure**.**addChain**(**chain**);**

Figure 20 - A chain containing global hinges with reference axes. Anticlockwise (red) and clockwise (blue) constraint angles are measured with respect to the hinge reference axis (cyan). On the left is close-up of a hinge, while the right images show an example solved configuration.



### 3D Demo 6 - Chains with local hinges

FabrikStructure3D structure **=** **new** FabrikStructure3D**();**

int numChains **=** 3**;**

float rotStep **=** 360.0f **/** **(**float**)**numChains**;**

Vec3f rotationAxis **=** **new** Vec3f**();**

Vec3f referenceAxis **=** **new** Vec3f**();**

Vec3f boneDirection **=** Z\_AXIS.negated**();** // i.e. into the screen

float boneLength **=** 10.0f**;**

// We'll create a circular arrangement of 3 chains with alternate bones each  
// constrained about different local axes. Note: Local hinge rotation axes are   
// relative to the generated rotation matrix of the previous bone in the chain.

**for** **(**int chainLoop **=** 0**;** chainLoop **<** numChains**;** **++**chainLoop**)** **{**

// Set colour and axes. Reference axes must be in the plane of hinge axes.

Colour4f chainColour **=** **new** Colour4f**();**

**switch** **(**loop **%** numChains**) {**

**case** 0**:**

chainColour **=** Utils**.**RED**;**

rotationAxis **=** **new** Vec3f**(**X\_AXIS**);**

referenceAxis **=** **new** Vec3f**(**Y\_AXIS**);**

**break;**

**case** 1**:**

chainColour **=** Utils**.**GREEN**;**

rotationAxis **=** **new** Vec3f**(**Y\_AXIS**);**

referenceAxis **=** **new** Vec3f**(**X\_AXIS**);**

**break;**

**case** 2**:**

chainColour **=** Utils**.**BLUE**;**

rotationAxis **=** **new** Vec3f**(**Z\_AXIS**);**

referenceAxis **=** **new** Vec3f**(**Y\_AXIS**);**

**break;**

**}**

// Create a new chain and set the basebone start and end locations

FabrikChain3D chain **=** **new** FabrikChain3D**();**

Vec3f start **=** **new** Vec3f**(**0.0f**,** 0.0f**,** **-**40.0f**);**

start **=** Vec3f**.**rotateYDegs**(**start**,** rotStep **\*** **(**float**)**chainLoop**);**

Vec3f end **=** start**.**plus**(** boneDirection**.**times**(**boneLength**)** **);**

// ...then create a base bone, set its colour, and add it to the chain.

FabrikBone3D basebone **=** **new** FabrikBone3D**(**startLoc**,** endLoc**);**

basebone**.**setColour**(**chainColour**);**

chain**.**addBone**(**basebone**);**

// Add alternating local hinge and unconstrained bones to the chain

float constraintDegs **=** 90.0f**;**

**for** **(**int boneLoop **=** 0**;** boneLoop **<** 6**;** **++**boneLoop**) {**

**if** **(**boneLoop **%** 2 **==** 0**) {**

// EITHER: For no reference constraints:

chain**.**addConsecutiveFreelyRotatingHingedBone**(**boneDirection**,** boneLength**,** JointType**.**LOCAL\_HINGE**,** hingeAxis**,** Utils**.**GREY**);**

// OR: To apply reference constraints:

chain**.**addConsecutiveHingedBone**(**boneDirection**,** boneLength**,**   
 JointType**.**LOCAL\_HINGE**,** rotationAxis**,** constraintDegs**,**  constraintDegs**,** referenceAxis**,** Utils**.**GREY**);**

**} else {**

chain**.**addConsecutiveBone**(**boneDirection**,** boneLength**,** chainColour**);**

**}**

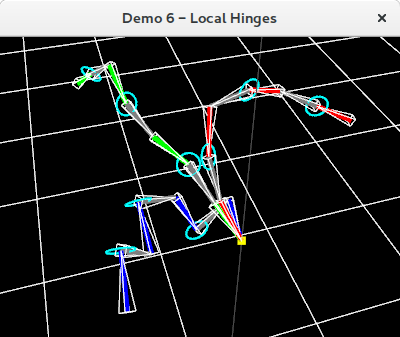
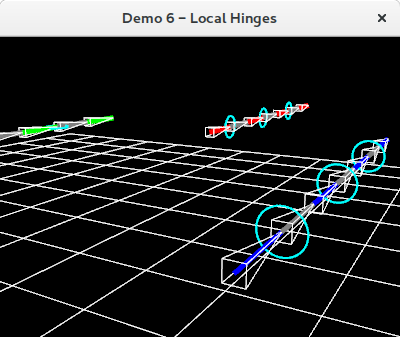
**}**

// Finally, add the chain to the structure

structure**.**addChain**(**chain**);**

**}**

Figure 21 - Chains with local hinges shown (left) in their initial configuration and (right) in an example solved configuration. Each local hinge's rotation axis and reference axis are converted into the coordinate system of the previous bone in the chain before being enforced.



### 3D Demo 7 - Connecting chains

FabrikStructure3D structure **=** **new** FabrikStructure3D**();**

FabrikChain3D chain **=** **new** FabrikChain3D**();**

Colour4f colour **=** **new** Colour4f**(**Utils**.**GREEN**);**

Vec3f boneDirection **=** Z\_AXIS.negated**();** // i.e. into the screen

float boneLength **=** 10.0f**;**

// Create a new chain, set basebone start and end locations then add to chain

Vec3f start **=** **new** Vec3f**(**0.0f**,** 0.0f**,** 40.0f**);**

Vec3f end **=** start**.**plus**(** boneDirection**.**times**(**boneLength**)** **);**

FabrikBone3D basebone **=** **new** FabrikBone3D**(**start**,** end**);**

basebone**.**setColour**(**colour**);**

chain**.**addBone**(**basebone**);**

// Add additional consecutive, unconstrained bones to the chain

**for** **(**int boneLoop **=** 0**;** boneLoop **<** 5**;** boneLoop**++) {**

colour **=** **(**boneLoop **%** 2 **==** 0**)** **?** colour**.**lighten**(**0.4f**)** **:** colour**.**darken**(**0.4f**);**

chain**.**addConsecutiveBone**(**defaultBoneDirection**,** defaultBoneLength**,** colour**);**

**}**

// Add the chain to the structure and create a second chain (base location isn't

// particularly important, it'll 'snap' to the connection point on the first chain)

structure**.**addChain**(**chain**);**

FabrikChain3D secondChain **=** **new** FabrikChain3D**();**

FabrikBone3D base **=** **new** FabrikBone3D**(** **new** Vec3f**(**100.0f**),** **new** Vec3f**(**110.0f**)** **);**

secondChain**.**addBone**(**base**);**

// We may optionally choose to constrain the chain we are connecting to the first // chain about a global rotor constraint, for example, 45 degrees about the X-axis.

secondChain**.**setRotorBaseboneConstraint**(**BaseboneConstraintType3D**.**GLOBAL\_ROTOR**,**  X\_AXIS**,** 45.0f**);**

secondChain**.**addConsecutiveBone**(**X\_AXIS**,** 20.0f**);**

secondChain**.**addConsecutiveBone**(**Y\_AXIS**,** 20.0f**);**

secondChain**.**addConsecutiveBone**(**Z\_AXIS**,** 20.0f**);**

// Set the colour of all bones in the chain then connect it to the first chain...

// Params: chain we're connecting, host chain number, host bone number, conn. point

secondChain**.**setColour**(**Utils**.**RED**);**

structure**.**connectChain**(**secondChain**,** 0**,** 0**,** BoneConnectionPoint3D**.**START**);**

// We can keep adding the same chain at various points if we like, because the

// chain we connect is actually a clone of the one we provide, and not the   
// original 'secondChain' object

secondChain**.**setColour**(**Utils**.**WHITE**);**

// Basebone constraints may be set on the connecting chain as desired prior to   
// connection by using the following methods:

// For GLOBAL\_ROTOR:

secondChain**.**setRotorBaseboneConstraint**(**BaseboneConstraintType3D**.**GLOBAL\_ROTOR**,** X\_AXIS**,** 45.0f**);**

// For LOCAL\_ROTOR:

secondChain**.**setRotorBaseboneConstraint**(**BaseboneConstraintType3D**.**LOCAL\_ROTOR**,** X\_AXIS**,** 45.0f**);**

// For GLOBAL\_HINGE:

secondChain**.**setFreelyRotatingGlobalHingedBasebone**(**Y\_AXIS**);**

// For GLOBAL\_HINGE with reference constraints:

secondChain**.**setGlobalHingeBaseboneConstraint**(**Y\_AXIS**,** 90.0f**,** 45.0f**,** X\_AXIS**);**

// For LOCAL\_HINGE:

secondChain**.**setFreelyRotatingLocalHingedBasebone**(**Y\_AXIS**);**

// For LOCAL\_HINGE with reference constraints:

secondChain**.**setLocalHingeBaseboneConstraint**(**Y\_AXIS**,** 90.0f**,** 45.0f**,** X\_AXIS**);**

// Once any basebone constraints have been applied, we're free to add the chain to   
// the structure - in this case we're connecting our secondChain to the end of bone   
// 1 in chain 0.

structure**.**connectChain**(**secondChain**,** 0**,** 1**,** BoneConnectionPoint3D**.**END**);**

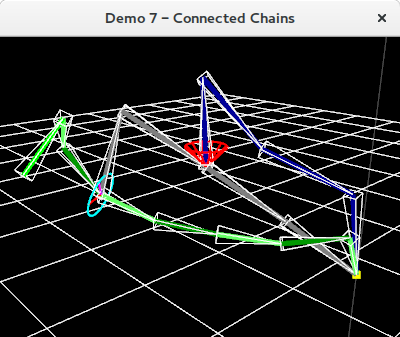


Figure 22 - A chain with a global rotor basebone constraint (in blue) attached to a chain with a local hinge with reference axis basebone constraint (in grey) attached to a chain without any basebone constraint (in green).

### 3D Demo 12 – Connected chains with embedded targets

**Note**: We jump from demos 7 to 12 as 8 through 11 show variations on a theme

mStructure **=** **new** FabrikStructure3D**(**demoName**);**

Colour4f boneColour **=** **new** Colour4f**(**Utils**.**GREEN**);**

// Create a new chain...

FabrikChain3D chain **=** **new** FabrikChain3D**();**

// ...then create a basebone, set its draw colour and add it to the chain.

Vec3f startLoc **=** **new** Vec3f**(**0.0f**,** 0.0f**,** 40.0f**);**

Vec3f endLoc **=** startLoc**.**plus**(** defaultBoneDirection**.**times**(**defaultBoneLength**)** **);**

FabrikBone3D basebone **=** **new** FabrikBone3D**(**startLoc**,** endLoc**);**

basebone**.**setColour**(**boneColour**);**

chain**.**addBone**(**basebone**);**

// Add additional consecutive, unconstrained bones to the chain

**for** **(**int boneLoop **=** 0**;** boneLoop **<** 7**;** boneLoop**++)**

**{**

boneColour **=** **(**boneLoop **%** 2 **==** 0**)** **?** boneColour**.**lighten**(**0.4f**)** **:**  boneColour**.**darken**(**0.4f**);**

chain**.**addConsecutiveBone**(**defaultBoneDirection**,** defaultBoneLength**,** boneColour**);**

**}**

// Finally, add the chain to the structure

mStructure**.**addChain**(**chain**);**

// Create a second chain which will be connected to the first and will use an   
// embedded target (specified in the main loop)

FabrikChain3D secondChain **=** **new** FabrikChain3D**(**"Second Chain"**);**

secondChain**.**setEmbeddedTargetMode**(true);**

FabrikBone3D base **=** **new** FabrikBone3D**(** **new** Vec3f**(),** **new** Vec3f**(**15.0f**,** 0.0f**,** 0.0f**)** **);**

secondChain**.**addBone**(**base**);**

// Set this second chain to have a freely rotating global hinge which rotates about   
// the Y axis. Note: We MUST add the basebone to the chain before we can set the   
// basebone constraint on it.

secondChain**.**setHingeBaseboneConstraint**(**BaseboneConstraintType3D**.**GLOBAL\_HINGE**,** Y\_AXIS**,** 90.0f**,** 45.0f**,** X\_AXIS**);**

// Add some additional bones

secondChain**.**addConsecutiveBone**(**X\_AXIS**,** 20.0f**);**

secondChain**.**addConsecutiveBone**(**X\_AXIS**,** 20.0f**);**

secondChain**.**addConsecutiveBone**(**X\_AXIS**,** 20.0f**);**

secondChain**.**setColour**(**Utils**.**GREY**);**

// Connect this second chain to the start point of bone 3 in chain 0 of the struct

mStructure**.**connectChain**(**secondChain**,** 0**,** 3**,** BoneConnectionPoint3D**.**START**);**

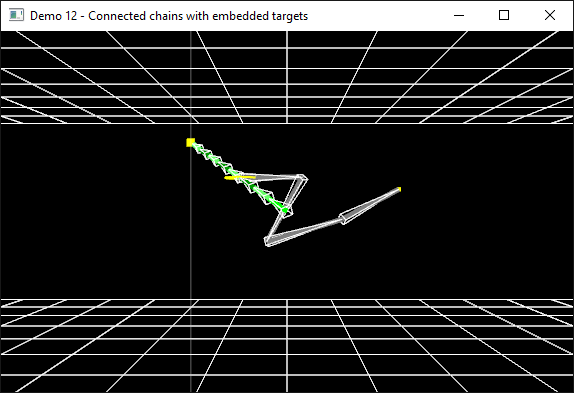


Figure 23 - Connected chains with embedded target locations (i.e. multiple end-effectors in a single structure). When embedded target mode is enabled for a chain then solving that chain uses the embedded target rather than any provided target location.