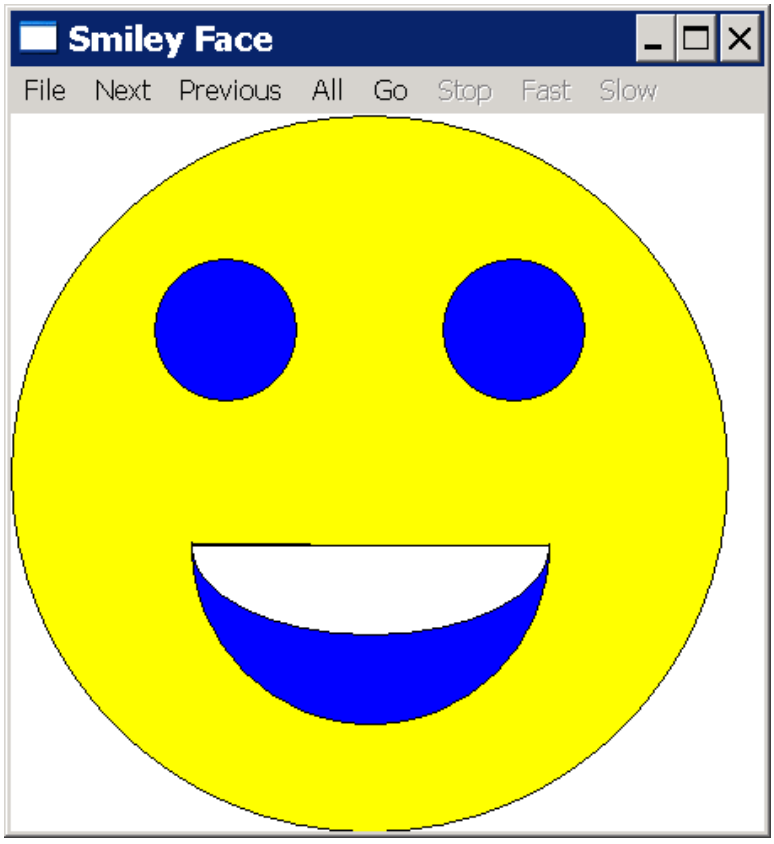
SoftIntegration®

QuickAnimation™

Version 8.0

User's Guide



How to Contact SoftIntegration

Mail	SoftIntegration, Inc. 216 F Street, #68 Davis, CA 95616
Phone	+ 1 530 297 7398
Fax	+ 1 530 297 7392
Web	http://www.softintegration.com
Email	info@softintegration.com

November, 2016

Table of Contents

Chapter 1

QuickAnimation for Display and Animation of Objects

1.1 Introduction

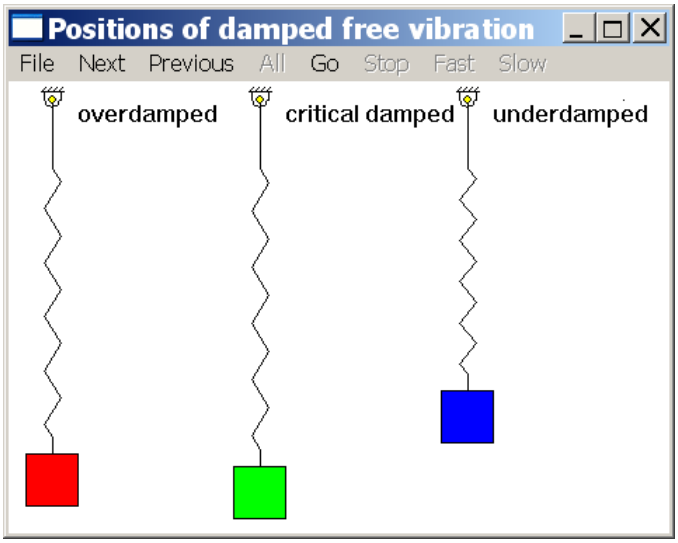


Figure 1.1: QuickAnimation window showing the positions of three vibration systems.

QuickAnimation^{TM} is a program for quick animation and display of various objects, based on specified x-y coordinate data. Like other SoftIntegration products, simplicity and easy to use is the key for this utility program. QuickAnimation^{TM} is especially suitable for animation of two-dimensional mechanical systems. For example, the QuickAnimation^{TM} window shown in Figure 1.1 displays a menu bar and overdamped, critical damped, and underdamped vibration systems. The displayed objects are drawn in the largest area of the window with a title above it. The details on how to use the QuickAnimation^{TM} program of **qanimate** for animation are described in this chapter.

1.2 User Interface for QuickAnimation™

A QuickAnimation™ data file for objects typically has a file extension .qnm. The data file can be invoked by the QuickAnimation™ command **qanimate** as follows:

```
qanimate datafile.qnm
```

to launch the QuickAnimation™ window, as shown in Figure 1.1.

The menu bar in the QuickAnimation window contains a series of menus which manipulate the animated system. The `File` menu allows one to quit the program. The `Next` and `Prev` buttons control the frame of an animation, and the `All` button displays all frames at once. The `Fast` and `Slow` buttons change the speed of animation. The `Go` and `Stop` buttons start and stop animation, respectively. The system can move in either direction by pressing the `Prev` button for one direction and the `Next` button for the opposite direction. When the `Go` button is pressed, the system will move in the direction previously assigned by the `Prev` or `Next` button.

1.3 Input Data Format

The typical format for a QuickAnimation data file is displayed in Figure 1.2. It is specified with the following typographical notation:

- `Typewriter` text specifies actual keywords.
- *`Emphasized`* text is specified by the user.
- Text between square brackets '`[]`' are optional.
- The line character '`|`' specifies an "OR" condition.

The character '`#`' on the first line delimits a comment. QuickAnimation will ignore anything on that line following the '`#`' character. The title of the mechanical system is specified by the `title` keyword followed by the title string delimited by the double quotation character, '`"`'. Keyword `fixture` allows the following lines to define the fixed objects. The `primitives` are commands used to define general and mechanical components of the displayed or animated system. The `animate` begins the inputting of of data for animation. Each line following keyword `animate` represents one frame of the animation, as indicated by the superscript on `primitive`. The option `restart` specifies that when the animation is finished, it starts again from the beginning. The option `reverse` specifies that when the animation is finished, it starts the animatin backwards from the last frame till the first frame. The `primitives` following the keyword `stopped` will be displayed only when animation is stopped. A frame consists of a data set that can contain multiple primitives. The continuation character `'\'` can be used to span primitives in a data set over multiple lines. An animation with n number of frames requires n number of data sets.

```
# comment
title "title string"
fixture
primitives
animate [ restart | reverse ]
primitives^1 [ stopped primitives^1 ]
primitives^2 [ stopped primitives^2 ]

                .
                .
                .

primitives^n [ stopped primitives^n ]
```

Figure 1.2: QuickAnimation data format.

1.3.1 General Drawing Primitives

Figure 1.3 shows the various general drawing primitives available for QuickAnimation. These primitives allow for the drawing of an arc, line, segment, circle, polygon, and rectangle as well as the insertion of text into a QuickAnimation program. The syntax for drawing such primitives are displayed in Figure 1.4. As an example, consider the sytax for drawing a line. One may specify a line by typing `line` followed by the x- and y-coordinates of the starting and ending points of the line (i.e. `line 0 0 2 3` draws a line from the origin to point (2,3) in the Cartesian coordinate system). Multiple lines may be linked together by adding more coordinate points after the `line` statement. Similarly, a circle may be drawn by specifying its center point and radius according to the syntax in Figure 1.4. The various options available for each general drawing primitives are displayed in Figure 1.5, and an example of color and font options is listed in Figure 1.6.

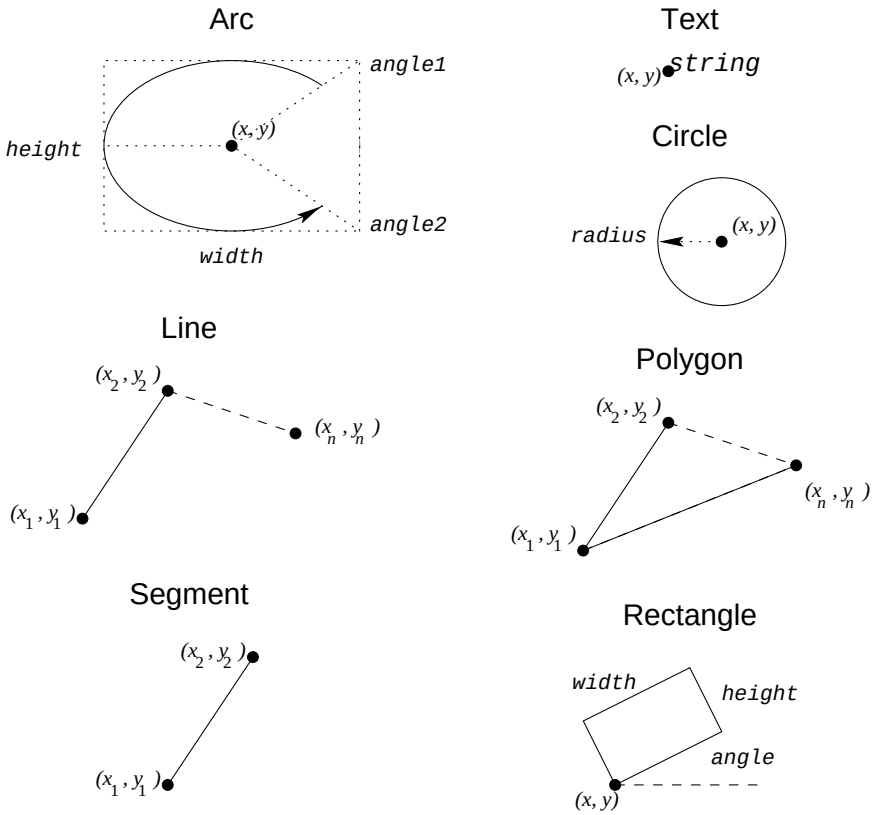


Figure 1.3: Graphical representation of general drawing primitives

```
line x1 y1  x2 y2  [...  xn yn ]
arc x y width height angle1 angle2
segment x1 y1 x2 y2
rectangle x y width height [ angle angle ]
polygon x1 y1  x2 y2  x3 y3  ...  xn yn
text x y string
circle x y radius
dot x y
```

Figure 1.4: Syntax for general drawing primitives

```
line
segment   ...
          [ pen color ]
          [ linewidth pixelwidth ]
          [ linestyle solid |
                      dashed [ length pixellength ] |
                      dotted [ gap pixelgap ] ]
          [ capstyle butt | round | projecting ]
          [ joinstyle miter | round | bevel ]
          [ depth depth ]
arc
circle
polygon
rectangle ...
          [ pen color ]
          [ fill color [ intensity percent ]
                       [ pattern number ] ]
          [ linewidth pixelwidth ]
          [ linestyle solid |
                      dashed [ length pixellength ] |
                      dotted [ gap pixelgap ] ]
          [ capstyle butt | round | projecting ]
          [ joinstyle miter | round | bevel ]
          [ depth depth ]
text      ...
          [ pen color ]
          [ depth depth ]
          [ font fontname ]
dot       ...
          [ pen color ]
          [ depth depth ]
```

Figure 1.5: Options for general drawing primitives

```
...  color { red | blue | yellow | white | black | grey90 ...  }
...  font { fixed | 6x13 | 6x13bold | lucidasanstypewriter-12 ...
}
```

Figure 1.6: Sample color and font options

Colors and fonts are specified by the X Window System in Unix and Windows.
The valid color names and their corresponding GRB values are listed below.

```
 Color Name         Hexadecimal R   G    B
                                values
-----------------------------------------
 white               #ffffff = 255 255 255
 black               #000000 =   0   0   0
 gray0               #000000 =   0   0   0
```

```
grey0              #000000 =   0   0   0
gray10             #1a1a1a =  26  26  26
grey10             #1a1a1a =  26  26  26
gray20             #333333 =  51  51  51
grey20             #333333 =  51  51  51
gray30             #4d4d4d =  77  77  77
grey30             #4d4d4d =  77  77  77
gray40             #666666 = 102 102 102
grey40             #666666 = 102 102 102
gray50             #7f7f7f = 127 127 127
grey50             #7f7f7f = 127 127 127
gray60             #999999 = 153 153 153
grey60             #999999 = 153 153 153
gray70             #b3b3b3 = 179 179 179
grey70             #b3b3b3 = 179 179 179
gray80             #cccccc = 204 204 204
grey80             #cccccc = 204 204 204
gray90             #e5e5e5 = 229 229 229
grey90             #e5e5e5 = 229 229 229
gray100            #ffffff = 255 255 255
grey100            #ffffff = 255 255 255
gray               #bebebe = 190 190 190
grey               #bebebe = 190 190 190
light-gray         #d3d3d3 = 211 211 211
light-grey         #d3d3d3 = 211 211 211
dark-gray          #a9a9a9 = 169 169 169
dark-grey          #a9a9a9 = 169 169 169
red                #ff0000 = 255   0   0
light-red          #f03232 = 240  50  50
dark-red           #8b0000 = 139   0   0
yellow             #ffff00 = 255 255   0
light-yellow       #ffffe0 = 255 255 224
dark-yellow        #c8c800 = 200 200   0
green              #00ff00 =   0 255   0
light-green        #90ee90 = 144 238 144
dark-green         #006400 =   0 100   0
spring-green       #00ff7f =   0 255 127
forest-green       #228b22 =  34 139  34
sea-green          #2e8b57 =  46 139  87
blue               #0000ff =   0   0 255
light-blue         #add8e6 = 173 216 230
dark-blue          #00008b =   0   0 139
midnight-blue      #191970 =  25  25 112
navy               #000080 =   0   0 128
medium-blue        #0000cd =   0   0 205
royalblue          #4169e1 =  65 105 225
skyblue            #87ceeb = 135 206 235
cyan               #00ffff =   0 255 255
```

```
light-cyan          #e0ffff = 224 255 255
dark-cyan           #008b8b =   0 139 139
magenta             #ff00ff = 255   0 255
light-magenta       #f055f0 = 240  85 240
dark-magenta        #8b008b = 139   0 139
turquoise           #40e0d0 =  64 224 208
light-turquoise     #afeeee = 175 238 238
dark-turquoise      #00ced1 =   0 206 209
pink                #ffc0cb = 255 192 203
light-pink          #ffb6c1 = 255 182 193
dark-pink           #ff1493 = 255  20 147
coral               #ff7f50 = 255 127  80
light-coral         #f08080 = 240 128 128
orange-red          #ff4500 = 255  69   0
salmon              #fa8072 = 250 128 114
light-salmon        #ffa07a = 255 160 122
dark-salmon         #e9967a = 233 150 122
aquamarine          #7fffd4 = 127 255 212
khaki               #f0e68c = 240 230 140
dark-khaki          #bdb76b = 189 183 107
goldenrod           #daa520 = 218 165  32
light-goldenrod     #eedd82 = 238 221 130
dark-goldenrod      #b8860b = 184 134  11
gold                #ffd700 = 255 215   0
beige               #f5f5dc = 245 245 220
brown               #a52a2a = 165  42  42
orange              #ffa500 = 255 165   0
dark-orange         #ff8c00 = 255 140   0
violet              #ee82ee = 238 130 238
dark-violet         #9400d3 = 148   0 211
plum                #dda0dd = 221 160 221
purple              #a020f0 = 160  32 240
```

1.3.2 Processing qnm Files

A QuickAnimation file with the file extension `.qnm` can be edited in ChIDE with syntqax highlighting, as shown in Figure 1.7 for the QuickAnimation file `circles.qnm`. The animation for `circles.qnm` can be created by clicking the command `Animate->qanimate (Animate a qnm file)` or the function key F10, as shown in Figure 1.7. Figure 1.8 shows all frames for the animation.

The QuickAnimation file `circles.qnm` can also be processed by the command

```
qanimate circles.qnm
```

where the command `qanimate` is available in Ch.

1.3.3 Writing Programs for Quick Animation

A program can be written to generate the standard output in the QuickAnimation format, using the standard C functions such as **printf**(). The standard output can be sent to the QuickAnimation program **qanimate** directly in ChIDE by clicking the command `Animate->Output to qanimate` or the function key F11,

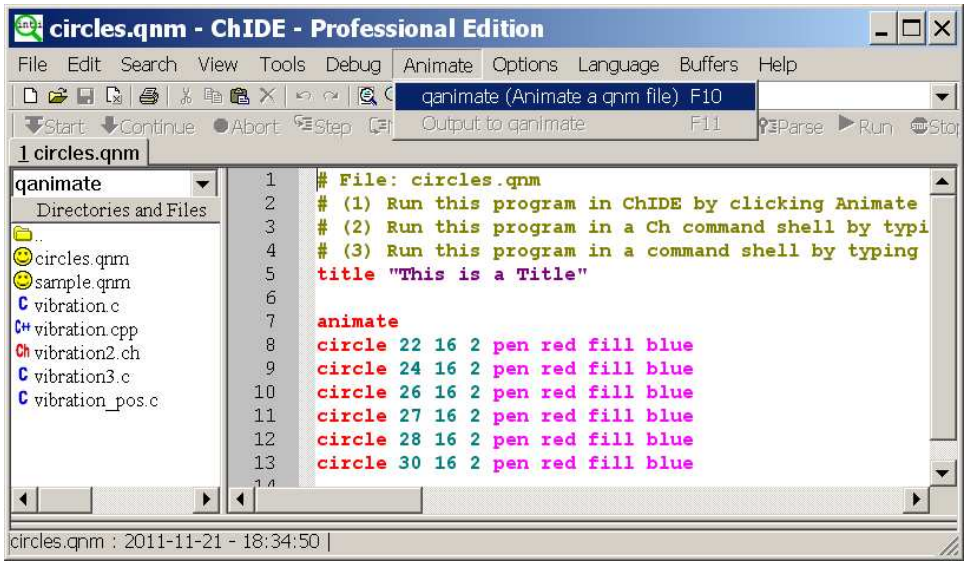


Figure 1.7: Executing a QuickAnimation file *circles.qnm*.

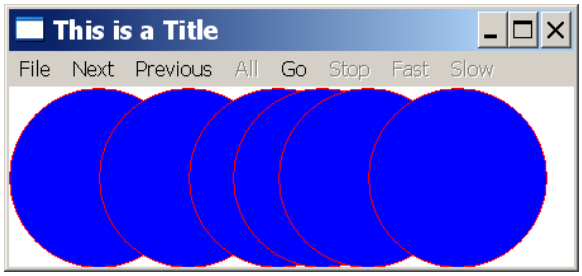


Figure 1.8: The output from executing the QuickAnimation file *circles.qnm* in Figure 1.7.

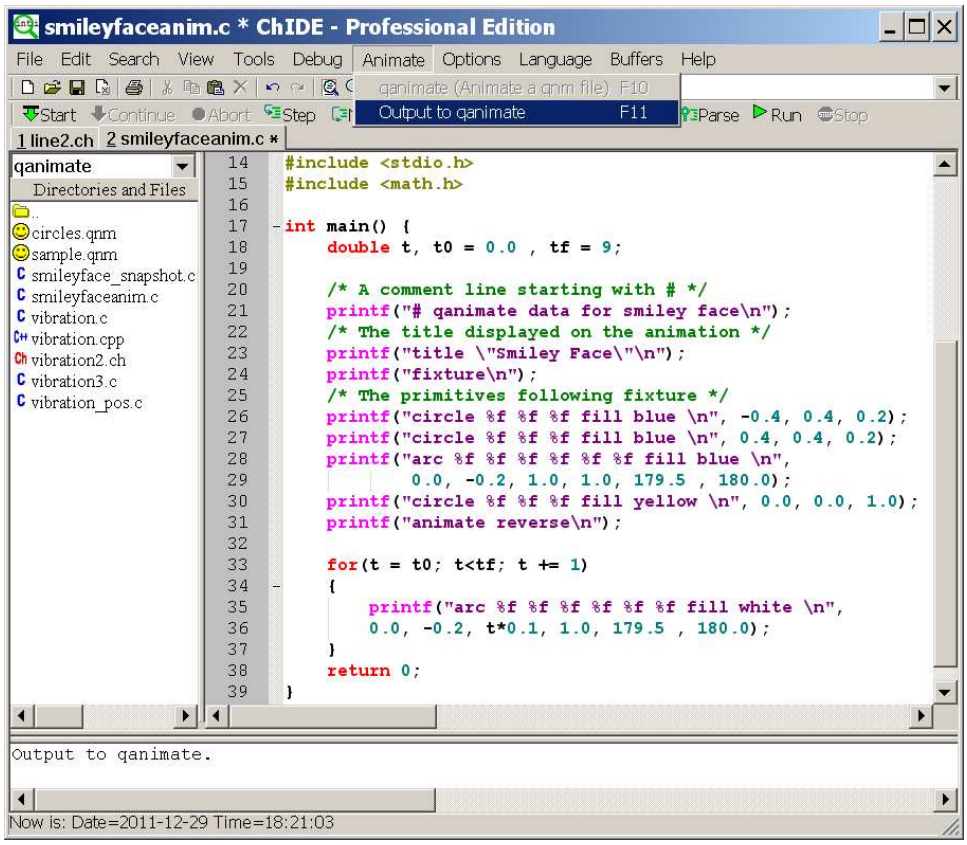


Figure 1.9: Executing a program with the standard output (stdout) sent to QuickAnimation.

as shown in Figure 1.9 for running the program `CHHOME/demos/qanimate/smileyfaceanim.c`. Figure 1.10 shows a snapshot of the generated animation.

The animation can also be created by typing the following commands in a Ch command shell.

```
smileyfaceanim.c | qanimate
```

or

```
smileyfaceanim.c > tmp1.qnm
qanimate tmp1.qnm
```

Figure 1.10: The output from executing the program *smileyfaceanim.c* in Figure 1.9.

1.3.4 Mechanical Drawing Primitives

The mechanical drawing primitives were built into QuickAnimation for ease of creating typical mechanical components, such as the springs and joints, of a mechanical system. The mechanical drawing primitives available in QuickAnimation are derived from the general drawing primitives. For example, a link is a combination of two circles connected by a line. All the available mechanical drawing primitives are shown in Fig. 1.11. These primitives are the primary tools used for creating animations of mechanical systems.

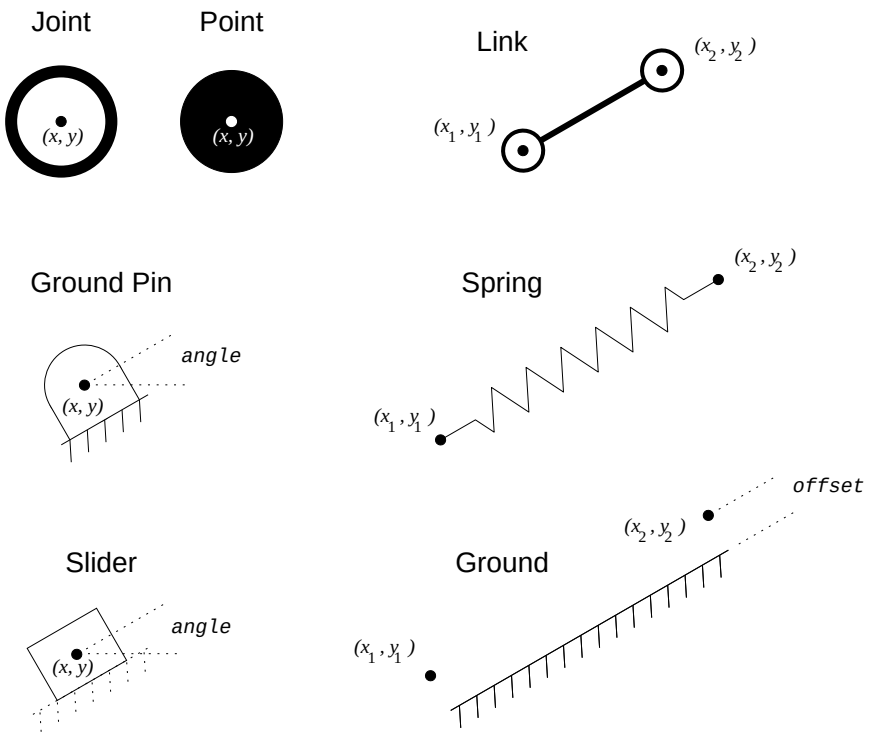


Figure 1.11: Graphical representation of mechanical drawing primitives

Point and Joint

The `point` primitive is basically circle with a filled-in center. It is usually used to emphasize a point on a mechanical system. The general syntax for a `point` is

`point` $x_1\ y_1$ `[`$x_2\ y_2$ `...` $x_n\ y_n$`] [trace],`

where $x_1y_1...x_ny_n$ specify the coordinate(s) of the joint(s), `trace` is an optional parameter used to specify whether the point is to be traced during animation. For example, to create a point at coordinate (1,3) with a trace the following command would be required:

```
point 1 3 trace.
```

Primitive `joint` is very similar to `point`. It is syntactically the same as the `point` primitive, but is comprised of a circle that is not filled-in. The joint represents a connection between two links or other mechanical components.

Link

As previously mentioned, the `link` primitive is a mechanical component formed by two circle primitives and a line primitive. This primitive is normally used for generating animations of mechanical linkages such as fourbar mechanisms. The general syntax for a `link` is given by the following:

`link` $x_1\ x_2\ \ x_2\ y_2$ `[...` $x_n\ y_n$`].`

The coordinates of the endpoints of the first link is specified by (x_1,y_1) and (x_2,y_2). Addition links may

be attached to the last link by indicating the coordinates of the links' other endpoints. A typical example of creating two links adjoined at a common endpoint would be

```
link 1 1  1 4  4 4
```

In this example, the endpoints of the first link are at coordinates (1,1) and (1,4). The second link is then attached to the first link at (1,4), and its other endpoint is located at (4,4). Note that the extra space between the endpoint coordinates are ignored during execution of the QuickAnimation program. They are present in the example to help distinguish the endpoints.

Ground

The `ground` primitive represents a reference area of the animation. It is stationary and fixed to its location. The syntax for `ground` is

`ground` x_1 y_1 x_2 y_2 `[offset` *pixeloffset*`]`
`[ticks` *forward* | *backward*`]`

For option `offset`, *pixeloffset* specifies the distance that the ground should be placed away for the x- and y-coordinates of the ground. Additionally, if the `ticks` option is used, and its value is `forward`, then the ground is specified as going from (x_1,y_1) to (x_2,y_2). Likewise, the opposite is true if the value of `ticks` is `backward`. The default value for option `ticks` is `forward`. For example,

```
ground 0 0  10 0 offset 2
```

will produce a `ground` section from x=0 to x=10 and two units below the line, y=0.

Ground Pin

In order to directly connect a mechanical system to `ground`, the `groundpin` primitive is used to specify the desired connection. The syntax for this primitive is given below as

`groundpin` *x y* `[angle` *angle*`]`

Coordinate (x,y) is the center point of the ground pin, and the optional argument `angle` *angle* describes the angular offset, in radians, relative to a horizontal position. In order to create a ground pin at the origin with a $45°$ rotational offset, the following statement should be declared:

```
groundpin 0 0  angle 45
```

Slider

The `slider` primitive is generated from the rectangle drawing primitive. It represents a block member of a mechanical system that is only capable of translation displacement. Similar to the ground pin, the slider can have an angular displacement that would allow it to translate on a sloped surface. Its syntax is defined as follow:

`slider` *x y* `[angle` *angle*`]`

As an example, consider a crank-slider mechanism that requires the slider to slide on a sloped surface, located at (3,4) that is about 30° relative to the ground. The slider portion of the mechanism can be created with the following statement:

```
slider 3 4 angle 30
```

Spring

The spring is a typical component of many mechanical systems. The availability of a `spring` primitive in QuickAnimation greatly increases the number of mechanical systems that can be modeled and animated. Its syntax is given as

`spring` $x_1\ y_1\ \ x_2\ y_2$,

where coordinates (x_1,y_1) and (x_2,y_2) specifies the endpoints of the spring. To create a spring from (1,1) to (3,5), the following should be entered in the QuickAnimation data file:

```
spring 1 1  3 5
```

1.4 Examples Using QuickAnimation™

The source code for these examples are distributed along with Ch. They can be found in the directory CHHOME/demos/qanimate, such as C:/Ch/demos/qanimate in Windows and /usr/local/ch/demos/qanimate in Unix. Additional examples including the source code can be found at http://www.softintegration.com/docs/ch/qanimate/.

1.4.1 Example 1: Data Format

The data file in Figure 1.12 illustrates how general and mechanical primitives are specified in a Quick-Animation file. Figure 1.13 shows the display when this data file is processed by the program **qanimate**.

```
# File: sample.qnm (this is a comment)
title "This is a Title"

fixture
#no fixture

animate

# low level primitives:
line 0 0 1 1.5 2 2 pen red \
    line 3 3 4 4
line 5 5 12 5 linestyle dashed length 2 pen green linewidth 1
line 5 6 12 6 linestyle dashed length 5 pen green linewidth 1
line 5 7 12 7 linestyle dotted gap 1 pen red linewidth 2
line 5 8 12 8 linestyle dotted gap 5 pen red linewidth 2
arc 11 11 4 4 0 270 fill grey90 linewidth 5
arc 12 12 10 11 0 90  13 13 5 5 0 360 linewidth 2 pen blue
segment 14 14 15 15  16 16 17 17 pen red
#color of text cannot be changed in Windows for now
text 18 5 string1 pen rgb:ffff/ffff/0
text 18 7 "This is a string2" pen red
text 18 9 "This is a string3" \
    font -*-lucidatypewriter-medium-*-*-*-12-*-*-*-*-*-*-*
circle 22 16 2 \
    stopped line 14 17 17 20 text 17.2 20 "center of circle"
rectangle 15 18 1 1 pen red fill grey
rectangle 17 20 2 1 angle 30

# higher linkage primitives
joint 18 18
point 19 19
link 20 20 21 21
groundpin 22 22 25 25 angle 30
link 22 22 25 25
polygon 4 10  5 10 6 13 3.5 14 fill green
spring 10 1  15 1
ground 17 1.0  19 2.0
# The traced trajectory shown on the upper left
point 0 20 trace
point 3 23 trace
point 6 25 trace
point 10 20 trace
```

Figure 1.12: A sample data file `sample.qnm`.

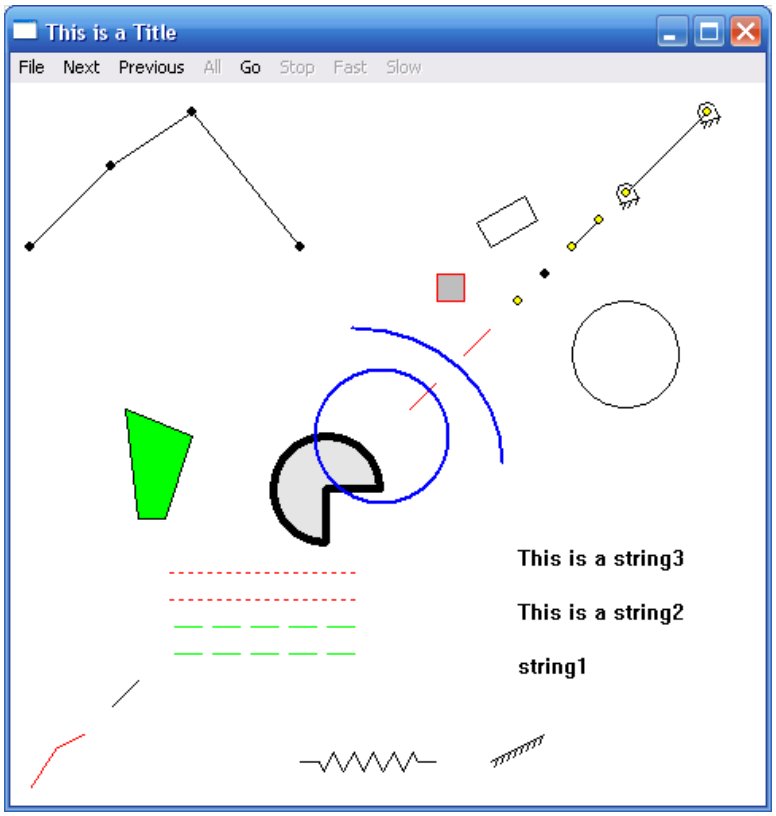


Figure 1.13: The QuickAnimation display based on the sample data file `sample.qnm`.

1.4.2 Example 2: Display Positions of Damped Free Vibrations

The QuickAnimation™ data file in Figure 1.14 can be used to display the vibration system shown in Figure 1.1. The first line starting with # is a comment line. Next, the title of the animation is set. Then, the fixtures are specified. Th first ground pin and joint are located in (0, 6) by the specification below.

```
groundpin 0 6 angle 180 joint 0 6
```

Th second one is located in (4, 6). Th third one is located in (8, 6). Three text strings for `overdamped`, `critical damped`, and `underdamped` are located next to the ground pins. Because the drawing area is calculated automatically based on the data for primitives without considering the text string width, the specification

```
dot 11 6 pen white
```

allows the text string `underdamped` displayed completely. The specification

```
rectangle -0.5 -0.818212 1 1 fill red \
spring 0 6 0 0.181788 \
```

draws the rectangle and spring for the overdamped vibration system. The next two continuation lines draw the rectangle and spring for the critical damped vibration system. The last two lines are for the underdamped vibration system. To display as fixed objects, these primitives could have been specified in multiple lines without continuation symbols.

```
# qanimate data for animation of vibration systems
title "Positions of damped free vibration"
fixture
groundpin 0 6 angle 180 joint 0 6
text 0.5 6 "overdamped"
groundpin 4 6 angle 180 joint 4 6
text 4.5 6 "critical damped"
groundpin 8 6 angle 180 joint 8 6
text 8.5 6 "underdamped"
dot 11 6 pen white
rectangle -0.5 -0.818212 1 1 fill red \
spring 0 6 0 0.181788 \
rectangle 3.5 -1.049575 1 1 fill green \
spring 4 6 4 -0.049575 \
rectangle 7.5 0.412908 1 1 fill blue \
spring 8 6 8 1.412908
```

Figure 1.14: The QuickAnimation™ file for vibration systems shown in Figure 1.1.

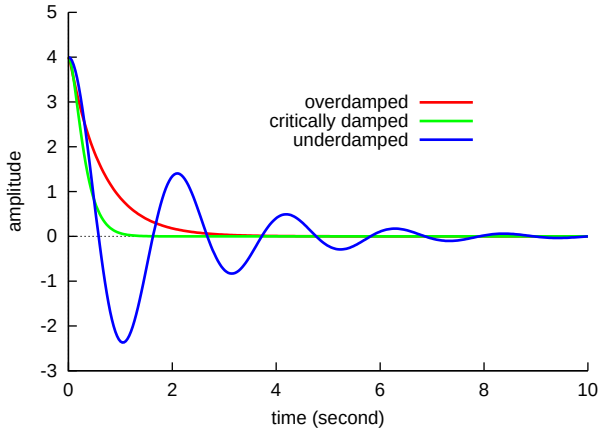


Figure 1.15: Three damped free vibrations.

```
/*********************************************************************
* File: vibration.cpp
* Display the positions of damped free vibrations of
* overdamped, critical damped, underdamped systems.
* Note: The details about this damped free vibration can be found in
* an exercise in Chapter 6 Functions in the book
* "C for Engineers and Scientists: An Interpretive Approach"
* by Harry H. Cheng, published by McGraw-Hill, 2009,
* ISBN: 0073376051, ISBN-13: 978-0073376059.
*********************************************************************/
#include <stdio.h>
#include <math.h>
#include <chplot.h>

/* The initial position of the vibration is 4.
   The initial velocity of the vibration is 0 */
double overdamped(double t) {
    return 4.12*exp(-1.57*t) - 0.12*exp(-54.2*t);
}

double criticaldamped(double t) {
    return 4*(1+6*t)*exp(-6*t);
}

double underdamped(double t) {
    return 4.06*exp(-0.5*t)*sin(3*t+1.4);
}

int main() {
    double t0, tf;
    int num = 100;       // number of points for plotting
    CPlot plot;

    t0 = 0;
    tf = 10;
    plot.title("Damped Free Vibration");
    plot.label(PLOT_AXIS_X, "time (second)");
    plot.label(PLOT_AXIS_Y, "x");
    plot.func2D(t0, tf, num, overdamped);
    plot.legend("overdamped", 0);
    plot.func2D(t0, tf, num, criticaldamped);
    plot.legend("critically damped", 1);
    plot.func2D(t0, tf, num, underdamped);
    plot.legend("underdamped", 2);
    plot.plotting();
    return 0;
}
```

Program 1: Program for creating displacement for damped vibration shown in Figure 1.15.

The three categories of the damped free vibrations, overdamped, critical damped, and underdamped, are described in Exercises in Chapter 6 Functions in the book *C for Engineers and Scientists: An Interpretive Approach* (by Harry H. Cheng, published by McGraw-Hill, 2009). Examples are given as follows.

1. **Overdamped**.
$$y_1(t) = 4.2e^{-1.57t} - 0.2e^{-54.2t}. \tag{1.1}$$
In this case, there is no oscillation. The motion decays and x approaches zero for large values of time as shown in Figure 1.15.

2. **Critically damped**.
$$y_2(t) = 4(1 - 3t)e^{-3t}. \tag{1.2}$$
The motion is also nonperiodic for a critically damped system. The mass will also reach the equilibrium position rapidly.

3. **Underdamped.**
$$y_3(t) = 4e^{-0.5t}\sin(3t + \pi/2). \tag{1.3}$$
In this case, there is oscillation as shown in Figure 1.15, which is created by Program 1. The solution is an exponentially decreasing harmonic function. However, because it is a damped motion, the body will eventually approach the equilibrium position for large values of time t.

Figure 1.1 shows the positions for the above three damped free vibration using Equations (1.1), (1.2), and (1.3) when time t is 2 seconds. Program 2 can be used to create the $\mathrm{QuickAnimation}^{\mathrm{TM}}$ data shown in Figure 1.14.

The displayed vibration systems in Figure 1.1, created by typing the following commands in a Ch command shell.

```
position.c | qanimate
```

or

```
position.c > tmp1.qnm
qanimate tmp1.qnm
```

```
/*********************************************************************
* File: position.c
* Display the position when t is 2 seconds for damped free vibration of
* overdamped, critical damped, underdamped systems.
* Run this program in Ch as follows:
*      position.c | qanimate
*                or
*      positions.c > tmp1.qnm
*      qanimate tmp1.qnm
* See CHHOME/docs/qanimate.pdf for detailed description of this program.
* Note: The details about this damped free vibration can be found in
* an exercise in Chapter 6 Functions in the book
* "C for Engineers and Scientists: An Interpretive Approach"
* by Harry H. Cheng, published by McGraw-Hill, 2009,
* ISBN: 0073376051, ISBN-13: 978-0073376059.
*********************************************************************/
#include <stdio.h>
#include <math.h>

/* The amplitude of the vibration is 4 */
double overdamped(double t) {
   return 4.2*exp(-1.57*t) - 0.2*exp(-54.2*t);
}

double criticaldamped(double t) {
   return 4*(1-3*t)*exp(-3*t);
}

double underdamped(double t) {
   return 4*exp(-0.5*t)*sin(3*t+M_PI/2);
}

int main() {
    double t, t0, tf;          // time
    double y1, y2, y3;         // displacement
    double pin1x = 0, pin1y=7, // pin 1
           pin2x = 4, pin2y=7, // pin 2
           pin3x = 8, pin3y=7; // pin 3

    /* A comment line starting with # */
    printf("# qanimate data for positions of vibration systems\n");
    /* The title displayed on the animation */
    printf("title \"Positions of damped free vibration\"\n");
    printf("fixture\n");
    /* The primitives following fixture */
    printf("groundpin %f %f angle 180 joint %f %f\n", pin1x, pin1y, pin1x, pin1y);
    printf("line %f %f %f %f\n", pin1x, pin1y, pin1x, pin1y-1 );
    printf("text %f %f \"overdamped\"\n", pin1x+0.5, pin1y);
    printf("groundpin %f %f angle 180 joint %f %f\n", pin2x, pin2y, pin2x, pin2y);
    printf("line %f %f %f %f\n", pin2x, pin2y, pin2x, pin2y-1 );
    printf("text %f %f \"critical damped\"\n", pin2x+0.5, pin2y);
    printf("groundpin %f %f angle 180 joint %f %f\n", pin3x, pin3y, pin3x, pin3y);
    printf("line %f %f %f %f\n", pin3x, pin3y, pin3x, pin3y-1 );
    printf("text %f %f \"underdamped\"\n", pin3x+0.5, pin3y);
    printf("dot 11 7 pen white\n"); // to display all text corretly

    t = 2; // 2 seconds
    y1 = overdamped(t);
    y2 = criticaldamped(t);
    y3 = underdamped(t);
    printf("rectangle %f %f %f %f fill red \\\n",-0.5, y1-1.0, 1.0, 1.0);
    printf("spring %f %f %f %f %f \\\n", pin1x, pin1y-1, pin1x, y1);
    printf("rectangle %f %f %f %f fill green \\\n", pin2x-0.5, y2-1.0, 1.0, 1.0);
    printf("spring %f %f %f %f %f \\\n", pin2x, pin2y-1, pin2x, y2);
    printf("rectangle %f %f %f %f fill blue \\\n", pin3x-0.5, y3-1.0, 1.0, 1.0);
    printf("spring %f %f %f %f %f \n", pin3x, pin3y-1, pin3x, y3);
    return 0;
}
```

Program 2: Program for creating QuickAnimation™ data file in Figure 1.14.

1.4.3 Example 3: Animation of Damped Free Vibrations

One may consider the motion of the above damped free vibration as an elevator approaching a stop. It would be very uncomfortable to ride if it were underdamped, and very slow to ride if it were overdamped. Critical damping provides the fastest and smoothest ride. The animation of the motion in QuickAnimation™ can be created by Program 3.

The result of Program 3 is shown in Figure 1.16. The data sets for creating the animation are generated by the code in a **for** loop. Figure 1.17 displays a snapshot of the QuickAnimation animation generated by the animation data file.

```
/*********************************************************************
* File: vibration.c
* Animate the damped free vibration of
* overdamped, critical damped, underdamped systems.
* The output is for animation coordinate data.
* (1a) Run this program in ChIDE by clicking Animate on the menu bar
* (1b) Run this program in Ch as follows:

*      vibration.c | qanimate
*                or
*      vibration.c > tmp1.qnm
*      qanimate tmp1.qnm
* (2) Click "Go" to view the animation
* See CHHOME/docs/qanimate.pdf for detailed description of this program.
* Note: The details about this damped free vibration can be found in
* an exercise in Chapter 6 Functions in the book
* "C for Engineers and Scientists: An Interpretive Approach"
* by Harry H. Cheng, published by McGraw-Hill, 2009,
* ISBN: 0073376051, ISBN-13: 978-0073376059.
*********************************************************************/
#include <stdio.h>
#include <math.h>

/* The initial position of the vibration is 4.
   The initial velocity of the vibration is 0 */
double overdamped(double t) {
    return 4.12*exp(-1.57*t) - 0.12*exp(-54.2*t);
}

double criticaldamped(double t) {
    return 4*(1+6*t)*exp(-6*t);
}

double underdamped(double t) {
    return 4.06*exp(-0.5*t)*sin(3*t+1.4);
}

int main() {
    double t, t0, tf;          // time
    double y1, y2, y3;         //displacement
    double pin1x = 0, pin1y=7, // pin 1
           pin2x = 4, pin2y=7, // pin 2
           pin3x = 8, pin3y=7; // pin 3

    /* A comment line starting with # */
    printf("# qanimate data for animation of vibration systems\n");
    /* The title displayed on the animation */
    printf("title \"Damped Free Vibration\"\n");
    printf("fixture\n");
    /* The primitives following fixture */
    printf("groundpin %f %f angle 180 joint %f %f\n", pin1x, pin1y, pin1x, pin1y);
    printf("line %f %f %f %f\n", pin1x, pin1y, pin1x, pin1y-1 );
    printf("text %f %f \"overdamped\"\n", pin1x+0.5, pin1y);
    printf("groundpin %f %f angle 180 joint %f %f\n", pin2x, pin2y, pin2x, pin2y);
    printf("line %f %f %f %f\n", pin2x, pin2y, pin2x, pin2y-1 );
    printf("text %f %f \"critical damped\"\n", pin2x+0.5, pin2y);
    printf("groundpin %f %f angle 180 joint %f %f\n", pin3x, pin3y, pin3x, pin3y);
    printf("line %f %f %f %f\n", pin3x, pin3y, pin3x, pin3y-1 );
    printf("text %f %f \"underdamped\"\n", pin3x+0.5, pin3y);
    printf("dot 11 7 pen white\n"); // to display all text corretly
    printf("animate restart\n");

    t0 = 0;
    tf = 10;
    for(t = t0; t<tf; t += 0.01) {
        y1 = overdamped(t);
        y2 = criticaldamped(t);
        y3 = underdamped(t);
        printf("rectangle %f %f %f %f fill red \\\n",-0.5, y1-1.0, 1.0, 1.0);
        printf("spring %f %f %f %f \\\n", pin1x, pin1y-1, pin1x, y1);
        printf("rectangle %f %f %f %f fill green \\\n", pin2x-0.5, y2-1.0, 1.0, 1.0);
        printf("spring %f %f %f %f \\\n", pin2x, pin2y-1, pin2x, y2);
        printf("rectangle %f %f %f %f fill blue \\\n", pin3x-0.5, y3-1.0, 1.0, 1.0);
        printf("spring %f %f %f %f \n", pin3x, pin3y-1, pin3x, y3);
    }
    return 0;
}
```

Program 3: Program to create the animation for damped free vibrations.

```
# qanimate data for animation of vibration systems
title "Positions of damped free vibration"
fixture
groundpin 0 6 angle 180 joint 0 6
text 0.5 6 "overdamped"
groundpin 4 6 angle 180 joint 4 6
text 4.5 6 "critical damped"
groundpin 8 6 angle 180 joint 8 6
text 8.5 6 "underdamped"
dot 11 6 pen white
rectangle -0.500000 -0.818212 1.000000 1.000000 fill red \
spring 0 6 0 0.181788 \
rectangle 3.500000 -1.049575 1.000000 1.000000 fill green \
spring 4 6 4 -0.049575 \
rectangle 7.500000 0.412908 1.000000 1.000000 fill blue \
spring 8 6 8 1.412908

    .
    .
    .

rectangle -0.500000 -0.999999 1.000000 1.000000 fill red \
spring 0.000000 6.000000 0.000000 0.000001 \
rectangle 3.500000 -1.000000 1.000000 1.000000 fill green \
spring 4.000000 6.000000 4.000000 -0.000000 \
rectangle 7.500000 -0.995843 1.000000 1.000000 fill blue \
spring 8.000000 6.000000 8.000000 0.004157
```

Figure 1.16: The QuickAnimation™ data generated by Program 3.

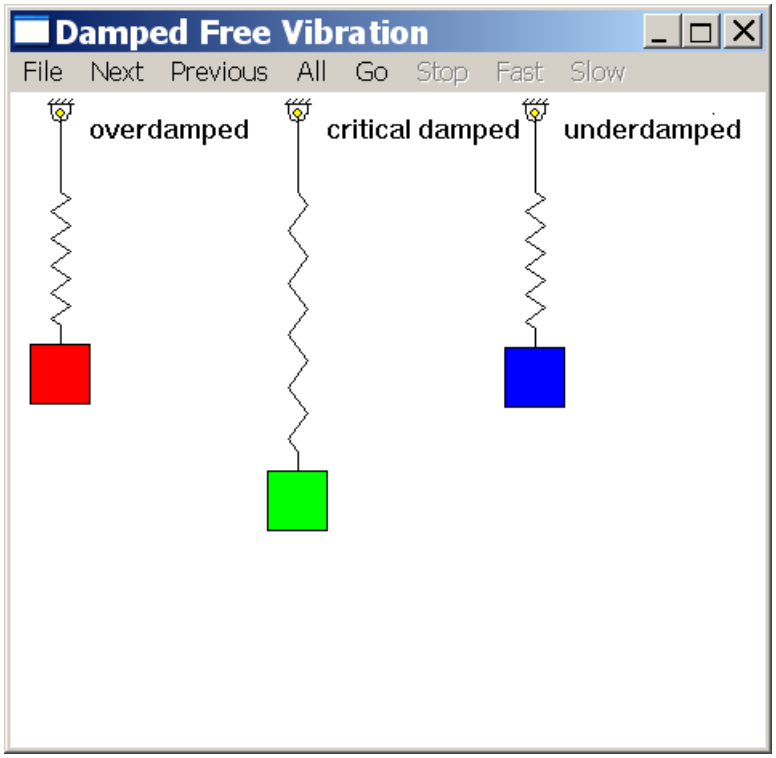


Figure 1.17: QuickAnimation window showing a snapshot of damped free vibrations.

Two additional program for animation of the damped free vibration using QuickAnimation animation are given below. They generate the same animation as that of the program `vibration.c` shown in Program 3. but with special handling of the animation data. Program `vibration2.ch` outputs the animation coordinate data to a temporary data file first. The animation data file is then processed by the command **qanimate**. After QuickAnimation™ is exited, the temporary animation data file is removed. In this case, the program `vibration2.ch` can readily run in ChIDE to create animation.

Program `vibration3.c` will also produce animation. The animation data are piped directly to the QuickAnimation program **qanimate** using the function **popen()** to automatically generate the animation. The command **qanimate** is not invoked as a command inside a Ch program. Execution of the program will simply generate the desired animation.

Listing of program `vibration2.ch`

```
/*********************************************************************
* File: vibration2.ch
* Animate the damped free vibration of
* overdamped, critical damped, underdamped systems.
* The output is for animation coordinate data.
* (1) Run this program in Ch.
* (2) Click "Go" to view the animation
* See CHHOME/docs/qanimate.pdf for detailed description of this program.
* Note: The details about this damped free vibration can be found in
* an exercise in Chapter 6 Functions in the book
* "C for Engineers and Scientists: An Interpretive Approach"
* by Harry H. Cheng, published by McGraw-Hill, 2009,
* ISBN: 0073376051, ISBN-13: 978-0073376059.
*********************************************************************/
#include <stdio.h>
#include <math.h>

/* The initial position of the vibration is 4.
   The initial velocity of the vibration is 0 */
double overdamped(double t) {
    return 4.12*exp(-1.57*t) - 0.12*exp(-54.2*t);
}

double criticaldamped(double t) {
    return 4*(1+6*t)*exp(-6*t);
}

double underdamped(double t) {
    return 4.06*exp(-0.5*t)*sin(3*t+1.4);
}

int main() {
    double t, t0, tf;          // time
    double y1, y2, y3;         // displacement
    double pin1x = 0, pin1y=7, // pin 1
           pin2x = 4, pin2y=7, // pin 2
           pin3x = 8, pin3y=7; // pin 3
    FILE *stream;
    char qnmFileName[1024];    // data file name for qanimate

    tmpnam(qnmFileName);
    stream = fopen(qnmFileName,"w");
    if (stream==NULL) {
        fprintf(stderr, "Error: cannot open '%s'\n", qnmFileName);
        exit(1);
    }

    /* The first line of the animation file must start with #qanimate */
    fprintf(stream, "# qanimate data for animation of vibration systems\n");
    /* The title displayed on the animation */
    fprintf(stream, "title \"Damped Free Vibration\"\n");
```

```
    fprintf(stream, "fixture\n");
    /* The primitives following fixture */
    fprintf(stream, "groundpin %f %f angle 180 joint %f %f\n", pin1x, pin1y, pin1x, pin1y);
    fprintf(stream, "line %f %f %f %f\n", pin1x, pin1y, pin1x, pin1y-1 );
    fprintf(stream, "text %f %f \"overdamped\"\n", pin1x+0.5, pin1y);
    fprintf(stream, "groundpin %f %f angle 180 joint %f %f\n", pin2x, pin2y, pin2x, pin2y);
    fprintf(stream, "line %f %f %f %f\n", pin2x, pin2y, pin2x, pin2y-1 );
    fprintf(stream, "text %f %f \"critical damped\"\n", pin2x+0.5, pin2y);
    fprintf(stream, "groundpin %f %f angle 180 joint %f %f\n", pin3x, pin3y, pin3x, pin3y);
    fprintf(stream, "line %f %f %f %f\n", pin3x, pin3y, pin3x, pin3y-1 );
    fprintf(stream, "text %f %f \"underdamped\"\n", pin3x+0.5, pin3y);
    fprintf(stream, "dot 11 7 pen white\n"); // to display all text corretly
    fprintf(stream, "animate restart\n");

    t0 = 0;
    tf = 10;
    for(t = t0; t<tf; t += 0.01) {
        y1 = overdamped(t);
        y2 = criticaldamped(t);
        y3 = underdamped(t);
        fprintf(stream, "rectangle %f %f %f %f fill red \\\n",-0.5, y1-1.0, 1.0, 1.0);
        fprintf(stream, "spring %f %f %f \\\n", pin1x, pin1y-1, pin1x, y1);
        fprintf(stream, "rectangle %f %f %f %f fill green \\\n", pin2x-0.5, y2-1.0, 1.0, 1.0);
        fprintf(stream, "spring %f %f %f \\\n", pin2x, pin2y-1, pin2x, y2);
        fprintf(stream, "rectangle %f %f %f %f fill blue \\\n", pin3x-0.5, y3-1.0, 1.0, 1.0);
        fprintf(stream, "spring %f %f %f %f \n", pin3x, pin3y-1, pin3x, y3);
    }
    fclose(stream);
    qanimate $qnmFileName
    remove(qnmFileName);
    return 0;
}
```

Listing of program `vibration3.c`

```
/*********************************************************************
* File: vibration3.c
* Animate the damped free vibration of
* overdamped, critical damped, underdamped systems.
* The output is for animation coordinate data.
* (1) Run this program in Ch.
* (2) Click "Go" to view the animation
* See CHHOME/docs/qanimate.pdf for detailed description of this program.
* Note: The details about this damped free vibration can be found in
* an exercise in Chapter 6 Functions in the book
* "C for Engineers and Scientists: An Interpretive Approach"
* by Harry H. Cheng, published by McGraw-Hill, 2009,
* ISBN: 0073376051, ISBN-13: 978-0073376059.
*********************************************************************/
#include <stdio.h>
#include <math.h>

/* The initial position of the vibration is 4.
   The initial velocity of the vibration is 0 */
double overdamped(double t) {
    return 4.12*exp(-1.57*t) - 0.12*exp(-54.2*t);
}

double criticaldamped(double t) {
    return 4*(1+6*t)*exp(-6*t);
}

double underdamped(double t) {
    return 4.06*exp(-0.5*t)*sin(3*t+1.4);
}

int main() {
    double t, t0, tf;          // time
```

```
    double y1, y2, y3;         // displacement
    double pin1x = 0, pin1y=7, // pin 1
           pin2x = 4, pin2y=7, // pin 2
           pin3x = 8, pin3y=7; // pin 3
    FILE *stream;

    stream = popen("qanimate","w");     // open qanimate pipe
    if (stream==NULL) {
        fprintf(stderr, "Error: popen() failed\n");
        exit(1);
    }

    /* A comment line starting with # */
    fprintf(stream, "# qanimate data for animation of vibration systems\n");
    /* The title displayed on the animation */
    fprintf(stream, "title \"Damped Free Vibration\"\n");
    fprintf(stream, "fixture\n");
    /* The primitives following fixture */
    fprintf(stream, "groundpin %f %f angle 180 joint %f %f\n", pin1x, pin1y, pin1x, pin1y);
    fprintf(stream, "line %f %f %f %f\n", pin1x, pin1y, pin1x, pin1y-1 );
    fprintf(stream, "text %f %f \"overdamped\"\n", pin1x+0.5, pin1y);
    fprintf(stream, "groundpin %f %f angle 180 joint %f %f\n", pin2x, pin2y, pin2x, pin2y);
    fprintf(stream, "line %f %f %f %f\n", pin2x, pin2y, pin2x, pin2y-1 );
    fprintf(stream, "text %f %f \"critical damped\"\n", pin2x+0.5, pin2y);
    fprintf(stream, "groundpin %f %f angle 180 joint %f %f\n", pin3x, pin3y, pin3x, pin3y);
    fprintf(stream, "line %f %f %f %f\n", pin3x, pin3y, pin3x, pin3y-1 );
    fprintf(stream, "text %f %f \"underdamped\"\n", pin3x+0.5, pin3y);
    fprintf(stream, "dot 11 7 pen white\n"); // to display all text corretly
    fprintf(stream, "animate restart\n");

    t0 = 0;
    tf = 10;
    for(t = t0; t<tf; t += 0.01) {
        y1 = overdamped(t);
        y2 = criticaldamped(t);
        y3 = underdamped(t);
        fprintf(stream, "rectangle %f %f %f %f fill red \\\n",-0.5, y1-1.0, 1.0, 1.0);
        fprintf(stream, "spring %f %f %f %f \\\n", pin1x, pin1y-1, pin1x, y1);
        fprintf(stream, "rectangle %f %f %f %f fill green \\\n", pin2x-0.5, y2-1.0, 1.0, 1.0);
        fprintf(stream, "spring %f %f %f %f \\\n", pin2x, pin2y-1, pin2x, y2);
        fprintf(stream, "rectangle %f %f %f %f fill blue \\\n", pin3x-0.5, y3-1.0, 1.0, 1.0);
        fprintf(stream, "spring %f %f %f %f \n", pin3x, pin3y-1, pin3x, y3);
    }
    pclose(stream);
    return 0;
}
```

Chapter 2

Web-Based Display and Animation of Objects

The QuickAnimation^{TM} program can be conveniently used to develop Web-based displayment of objects and animation.

2.0.1 Writing CGI Script Files

Processing data sent from an HTML document with the <FORM> tag requires a script file, which is located on the server side. Once data are passed to the server through Common Gateway Interface (CGI) or mechanisms, the necessary procedures are performed and the result is returned to the client. Details about CGI in Ch can be in *Ch CGI Toolkit User's Guide*. To use QuickAnimation^{TM} , the content type of the output form a CGI script need to be specified by the statement

```
Response.setContentType("application/x-qnm");
```

which indicates that the output is a QuickAnimation^{TM} application.

2.0.2 Configuration and Setup of Web Servers

In order to run the QuickAnimation^{TM} application from a Netscape Web server in a Unix operating system, the following line has to be added to the Netscape WWW server configuration file **mime.types** located in directory `server_home_dir/https-80_or_http/config`.

```
type=application/x-qnm     exts=qnm
```

For the Apache Web server, the line

```
application/x-qnm     qnm
```

may be added to file `server_home_dir/conf/mime.types`. Note that the Web server needs to restart in order for the changes to be effective.

Index