# Mathematical Double Pendulum for Android

**Thomas Müller**

Visualisierungsinstitut der Universität Stuttgart (VISUS), SFB716
Allmandring 19, 70569 Stuttgart, Germany

E-mail: `Thomas.Mueller@visus.uni-stuttgart.de`

**Abstract.** The mathematical double pendulum is an interesting exercise in a beginners' course of classical mechanics to determine equations of motion by means of the Euler-Lagrangian formalism. While the gravitational force is naturally fixed, an Android device offers the possibiliy to play around with a variable gravitational force direction depending on how the user holds its device with respect to actual gravity. In this short manuscript, the equations of motion for a variable gravitational force direction and velocity dependent frictional terms are deduced. Furthermore, some implementation details for the simulation and the visualization are presented.

Find the application in Google play store:

## 1. Introduction

The mathematical double pendulum consists of two masses $m_1$ and $m_2$. The first mass is fixed with a massless rod of length $l_1$ at the point 0, and the second mass is fixed with a massless rod of length $l_2$ at the first mass, see figure 1. If gravitation points in the negative $y$-direction, then 1 and 2 are the rest positions of both masses.

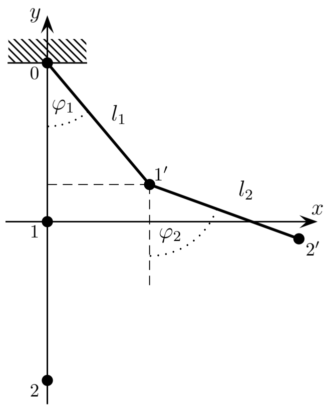

**Figure 1.** The mathematical double pendulum consists of two masses $m_1$ and $m_2$ and two massless rods of lengths $l_1$ and $l_2$, respectively.

In the following, we will determine the equations of motion for the mathematical double pendulum in two dimensions with friction and a variable direction of gravitation due to the orientation of the Android device.

## 2. Equations of motion in canonical coordinates

The equations of motion for the mathematical double pendulum can be deduced by means of the Euler-Lagrangian (EL) formalism with friction. Details to the EL formalism can be found in the standard literature to classical mechanics [1, 2].

The Cartesian coordinates of both masses can be easily read from figure 1, parametrized by the angles $\varphi_1$ and $\varphi_2$ measured from the negative $y$-axis,

$$x_1 = l_1 \sin\varphi_1, \quad y_1 = l_1 \cos\varphi_1, \tag{1}$$

$$x_2 = l_1 \sin\varphi_1 + l_2 \sin\varphi_2, \quad y_2 = l_1 \cos\varphi_1 + l_2 \cos\varphi_2. \tag{2}$$

The kinetic energy $T$ of the total system follows from

$$T = \frac{m_1}{2}\left(\dot{x}_1^2 + \dot{y}_1^2\right) + \frac{m_2}{2}\left(\dot{x}_2^2 + \dot{y}_2^2\right) \tag{3}$$

$$= \frac{m_1}{2} l_1^2 \dot{\varphi}_1^2 + \frac{m_2}{2}\left[l_1^2\dot{\varphi}_1^2 + l_2^2\dot{\varphi}_2^2 + 2l_1 l_2 \dot{\varphi}_1\dot{\varphi}_2 \cos(\varphi_1 - \varphi_2)\right], \tag{4}$$

where a dot means differentiation with respect to time, $\dot{x} = \mathrm{d}x/\mathrm{d}t$, and the derivatives of the coordinates read

$$\dot{x}_1 = l_1\dot{\varphi}_1 \cos\varphi_1, \quad \dot{y}_1 = -l_1\dot{\varphi}_1 \sin\varphi_1, \tag{5}$$

$$\dot{x}_2 = l_1\dot{\varphi}_1\cos\varphi_1 + l_2\dot{\varphi}_2\cos\varphi_2, \quad \dot{y}_2 = -l_1\dot{\varphi}_1\sin\varphi_1 - l_2\dot{\varphi}_2\sin\varphi_2. \tag{6}$$

The absolute values for the velocities are given by

$$v_1 = \sqrt{\dot{x}_1^2 + \dot{y}_1^2} = l_1\dot{\varphi}_1, \tag{7}$$

$$v_2 = \sqrt{\dot{x}_2^2 + \dot{y}_2^2} = \sqrt{l_1^2\dot{\varphi}_1^2 + l_2^2\dot{\varphi}_2^2 + 2l_1 l_2\dot{\varphi}_1\dot{\varphi}_2\cos(\varphi_1 - \varphi_2)}. \tag{8}$$

The potential energy $U$ depends on the direction of gravitation $\vec{g} = g(\sin\psi, \cos\psi)^T$, where $g$ is the absolute value of gravitation and $\psi$ is the direction measured in the same way as the angles $\varphi_i$. Hence,

$$U = -m_1\vec{g}\cdot\begin{pmatrix}x_1\\y_1\end{pmatrix} - m_2\vec{g}\cdot\begin{pmatrix}x_2\\y_2\end{pmatrix} \tag{9}$$

$$= -(m_1+m_2)gl_1\cos(\varphi_1 - \psi) - m_2 g l_2 \cos(\varphi_2 - \psi). \tag{10}$$

The equations of motion with friction follow from the Lagrangian function $L = T - U$ and the Euler-Lagrangian differential equations

$$0 = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial\dot{\varphi}_j} - \frac{\partial L}{\partial\varphi_j} - R_j \qquad \text{for} \quad j = \{1,2\}. \tag{11}$$

After a few lines of straightforward calculation, the Euler-Lagrangian equations yield

$$0 = \ddot{\varphi}_1 + \frac{g}{l_1}\sin(\varphi_1 - \psi) + \frac{m_2 l_2}{(m_1+m_2)l_1}\left[\cos(\varphi_1-\varphi_2)\ddot{\varphi}_2 + \sin(\varphi_1-\varphi_2)\dot{\varphi}_2^2\right] - R_1, \tag{12}$$

$$0 = \ddot{\varphi}_2 + \frac{g}{l_2}\sin(\varphi_2 - \psi) + \frac{l_1}{l_2}\left[\cos(\varphi_1-\varphi_2)\ddot{\varphi}_1 - \sin(\varphi_1-\varphi_2)\dot{\varphi}_1^2\right] - R_2. \tag{13}$$

In Cartesian coordinates, the frictional term $\vec{F}_i$ for the particle $i$ can be written as

$$\vec{F}_i = -h_i(v_i)\frac{\vec{v}_i}{v_i}, \qquad i = 1,\ldots,N, \tag{14}$$

where $h_i(v_i)$ is a function of the velocity $v_i$. In the canonical coordinates $\varphi_j$ used here, the frictional terms of equations (12) and (13) read

$$R_j = -\sum_{i=1}^{N} h_i(v_i)\frac{\partial v_i}{\partial\dot{\varphi}_j}. \tag{15}$$

Here, $N = 2$, and with $v_i^2 = \dot{x}_i^2 + \dot{y}_i^2$, these terms are given by

$$-R_1 = h_1(v_1)l_1 + h_2(v_2)\frac{1}{v_2}\left[l_1^2\dot{\varphi}_1 + l_1 l_2 \dot{\varphi}_2 \cos(\varphi_1 - \varphi_2)\right], \tag{16}$$

$$-R_2 = h_2(v_2)\frac{1}{v_2}\left[l_2^2\dot{\varphi}_2 + l_1 l_2 \dot{\varphi}_1 \cos(\varphi_1 - \varphi_2)\right]. \tag{17}$$

Equations (12) and (13) are not yet usable for standard numerical integration techniques, because the second derivatives are still coupled. But by replacing $\ddot{\varphi}_2$ of (13) into (12), and vice versa, the equations of motion read,

$$\ddot{\varphi}_1 = \frac{1}{1-\mu\cos^2(\varphi_1-\varphi_2)}\left\{-\frac{g}{l_1}\sin(\varphi_1-\psi)\right. \tag{18}$$
$$\left. -\frac{\mu l_2}{l_1}\left[\cos(\varphi_1-\varphi_2)\left[-\frac{g}{l_2}\sin(\varphi_2-\psi)+\frac{l_1}{l_2}\sin(\varphi_1-\varphi_2)\dot{\varphi}_1^2 - R_2\right] + \sin(\varphi_1-\varphi_2)\dot{\varphi}_2^2\right] + R_1\right\},$$

and

$$\ddot{\varphi}_2 = \frac{1}{1-\mu\cos^2(\varphi_1-\varphi_2)}\left\{-\frac{g}{l_2}\sin(\varphi_2-\psi)\right. \tag{19}$$
$$\left. +\frac{l_1}{l_2}\left[\cos(\varphi_1-\varphi_2)\left[\frac{g}{l_1}\sin(\varphi_1-\psi)+\frac{\mu l_2}{l_1}\sin(\varphi_1-\varphi_2)\dot{\varphi}_2^2 - R_1\right] + \sin(\varphi_1-\varphi_2)\dot{\varphi}_1^2\right] + R_2\right\}.$$

The mass relation $\mu = m_2/(m_1+m_2)$ is always less than 1 for finite masses.

## 3. Implementation details

Android apps are based on the Java [3] programming language with additional functionally for Android devices. An introduction to Android development and the API can be found at [4]. In the following, all parameters are given in SI units.

### *3.1. Pendulum simulation*

The equations of motion (EOMs), (19) and (20), of the double pendulum with friction and variable gravitational direction can be solved using the standard fourth-order Runge-Kutta integrator [5]. As the EOMs are independent of time, the Runge-Kutta formulas read

$$\mathbf{k_1} = h\mathbf{f}(y_n), \tag{20}$$

$$\mathbf{k_2} = h\mathbf{f}\left(y_n + \frac{1}{2}\mathbf{k_1}\right), \tag{21}$$

$$\mathbf{k_3} = h\mathbf{f}\left(y_n + \frac{1}{2}\mathbf{k_2}\right), \tag{22}$$

$$\mathbf{k_4} = h\mathbf{f}(y_n + \mathbf{k_3}), \tag{23}$$

$$\mathbf{y_{n+1}} = \mathbf{y_n} + \frac{1}{6}(\mathbf{k_1} + 2\mathbf{k_2} + 2\mathbf{k_3} + \mathbf{k_4}), \tag{24}$$

where $\mathbf{y_n}$ is the four-dimensional vector consisting of $\varphi_1$, $\varphi_2$, $\dot{\varphi}_1$, and $\dot{\varphi}_2$ at the current time step $n$. The function $\mathbf{f}$ follows from transforming the second-order ordinary differential equation into a first-order ODE. Thus, $\mathbf{f} = (\dot{\varphi}_1, \dot{\varphi}_2, \ddot{\varphi}_1, \ddot{\varphi}_2)$.

To solve the second order ODE, the initial positions $\varphi_i(t=0)$ and the initial velocities $\dot{\varphi}_i(t=0)$ have to be known. Here, the velocities are always set to zero and the initial positions can be set using the touch screen. As the simulation is driven by a 5 *ms* timer, the time step for the Runge-Kutta integration should be $h = 0.005$ for an approximate realtime animation.

## *3.2. Visualization*

The visualization of the double pendulum is based on the open graphics library for embedded systems (OpenGL ES 2.0) [6]. The graphics pipeline for object rendering consisting of a vertex and a fragment shader, see figure 2.

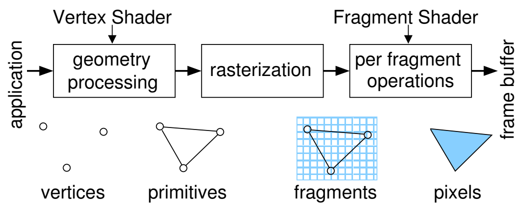

**Figure 2.** A minimal OpenGL rendering pipeline consists of a vertex and a fragment shader.

Here, the vertex shader is mainly used for orthographic projection and for scaling the arrow representing the gravitational force direction. It also generates the texture coordinates for the background and sets the pixel size of the pendulum bobs. The rest of the visualization is done in the fragment shader.

The background of the simulation domain is a checkerboard texture defined either with respect to Cartesian (`gridType==0`) or with respect to polar coordinates. Using the texture coordinates `texCoords` defined in the vertex shader, the resulting gray value follows from `val`,

```
float val = 0.0;
if (gridType==0) {
    val = 0.1 + 0.1*sign(sin(2.0*3.14159*texCoords.x)*sin(2.0*3.14159*texCoords.y));
} else {
    float r = length(texCoords);
    float phi = atan(texCoords.y,texCoords.x);
    val = 0.1 + 0.1*sign(sin(2.0*3.14159*r)*sin(12.0*phi));
}
```

The vertices for the arrow representing the gravitational force direction are scaled within the vertex shader, where `off` defines an offset and `wh` scales the input position `aPosition` of a vertex. These parameters are necessary, because the input positions for the top of the arrow are $(-1,0)$, $(1,0)$, $(0.1)$, and the positions of the leg read $(-1,0)$, $(1,0)$, $(-1,1)$, $(1,1)$. Additionally, any vertex is rotated by the angle `phi` to point into the direction of actual gravity.

```
vec2 vert = aPosition.xy*wh + off;
vert = mat2(cos(phi),-sin(phi),sin(phi),cos(phi)) * vert;
gl_Position = uMVPMatrix * vec4( vert, 0.0, 1.0 );
```

The shading for the pendulum bobs is realized within the fragment shader. As a bob is represented by a single point, its texture coordinates `tc` follow from the point coordinates `gl_PointCoord` which are determined within the fragment shader. Those parts that lie outside the radius 1 with respect to texture coordinates are discarded. A spherical appearance can be achieved by determing a fictitious normal direction `n` and calculating the dot product to the $z$-direction.

```
vec2 tc = 2.0*(gl_PointCoord - vec2(0.5));
if (length(tc)>1.0) {
    discard;
}
vec3 n = normalize(vec3(tc.x,tc.y,sqrt(1.0-length(tc))));
gl_FragColor = vec4(vec3(n.z)*0.5+0.5,1);
```

## 4. The App

Some screenshots of the *DoublePendulum* App are shown in figure 3. When starting the App, the pendulum is in its rest position where $\varphi_1 = \varphi_2 = 0$. The gravitation is fixed and points downwards. The pendulum parameters are $m_1 = m_2 = 1\ kg$, $l_1 = 1\ m$, $l_2 = 0.8\ m$, and there is no friction (damping).

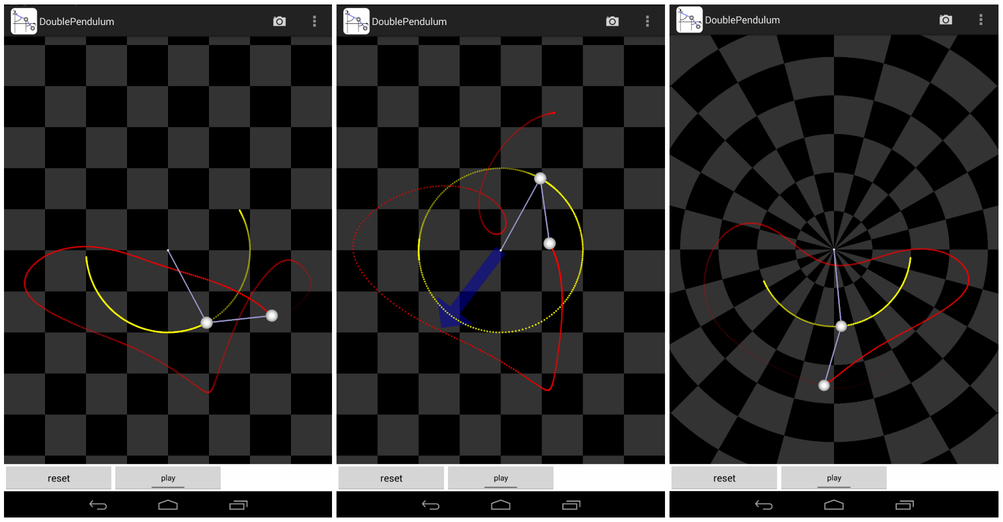

**Figure 3.** Screenshots of *DoublePendulum* with either Cartesian or polar grid as background. The blue arrow in the central image shows the direction and strength of gravitation.

To start the application, move the pendulum bobs using the touch screen and press the "play" button. For better control over the initial conditions, the initial angles $\varphi_i(t = 0)$ can be defined in the preference settings. Then, every time the "reset" button is pressed, these initial angles are used.

## Acknowledgments

This work was partially funded by Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Centre SFB 716.

## References

[1] Friedhelm Kuypers. *Klassische Mechanik*. VCH, 1993.
[2] Herbert Goldstein. *Classical Mechanics*. Addison-Wesley, 2001.
[3] The Java (SE) development kit can be downloaded from `http://www.oracle.com/technetwork/java/javase/downloads/index.html`.
[4] The developer site of android can be found at `http://developer.android.com/index.html`.
[5] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1994.
[6] Details to the Open Graphics Library (OpenGL) and the OpenGL Shading Language can be found at `http://www.opengl.org`.