Simulation of 2D physics of hand drawn objects using OpenCV and Box2D

Michal Sedlák [*]

[*] *Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Ilkovičova 3, 812 19 Bratislava, Slovakia (e-mail: michal.sedlak@stuba.sk)*

Abstract

Keywords: Maximum 5 keywords.

1. INTRODUCTION

This paper describes applying of Newtonian physics to hand drawn objects recognized in image from camera. Simulation of physics is used in many modern applications. You can find in implementations used by 3D drawing and animation programs, more complex used in game engines or exact and precise simulation in CAE programs. Paper describes process of animation of hand drawn object, from a capturing phase, over recognition of the objects, interpretation objects in physical engine, to animation of such objects. This approach can be applied in education of physics at elementary schools, with interactive blackboards, or in computer games.

2. OBJECT DETECTION AND OPEN COMPUTER VISION LIBRARY

To apply a physics to hand drawn objects we need to identify and isolate objects from image. We have used a web camera as a source and Open Computer Vision library as processing tool of the images.

2.1 OpenCV

In regards the book of Bradski and Kaehler (2008) OpenCV is a library for open source programming functions for real time computer vision, with more than five hundred optimized algorithms. It can be used with C++, C and Python. We chose Python for implementation in our application.

Simple image capture is shown in Listing 1.

```
1 self.camera = cv.CaptureFromCAM(-1)
2 self.image = cv.QueryFrame(self.camera)
3 self.DetectOutline(self.image)
```

Listing 1. Query image frame from web camera

In line 1 of listing 1 we initialize our web camera. In variable camera is allocated and initialized object that can query camera for new image. Then as we see in 2 we can get the image from camera and store it in the variable named image. Now when we have image data stored in the variable, we can process data to find outlines.

```
1  def DetectOutline(self, image):
2    image_size = cv.GetSize(image)
3    grayscale = cv.CreateImage(image_size, 8, 1)

4    cv.CvtColor(image, grayscale, cv.CV_BGR2GRAY)
5    cv.EqualizeHist(grayscale, grayscale)
6    storage = cv.CreateMemStorage(0)
7    cv.Threshold(grayscale, grayscale, 50, 255,
         cv.CV_THRESH_BINARY)
8    self.contours = cv.FindContours(grayscale,
9      cv.CreateMemStorage(),
10     cv.CV_RETR_TREE,
11     cv.CV_CHAIN_APPROX_SIMPLE)
12   if len(self.contours) > 0:
13     self.contours = cv.ApproxPoly (self.
          contours,
14       storage,
15       cv.CV_POLY_APPROX_DP,
16       1.5,
17       1)
18   return self.contours
```

Listing 2. Outline detection

In function in Listing 2 is shown how to find outlines of objects in image. We convert image to gray scale as seen on line 3. Then we run histogram equalization (line: 5). Equalization makes objects better visible and gives better output for thresholding (line: 7) which makes black and white image prepared for outline detection (line: 8). After outline detection we have tree of contours stored in the variable self.contours. These trees are iterable objects sorted from outer to inner outline connected by property h_next and v_next that we will describe in paragraph about creation of objects from outlines.

Contour can be very complicated and consist of thousands of points, which could cause too complicated objects. It is time demanding to simulate complicated objects, that is why we use polynomial approximation of the contour points. (line: 13).

Now we have all outlines stored in the outline tree structure, so we can create objects and apply a physics.

3. PHYSICS SIMULATION IN BOX2D

There is lot of physics engines that can be used for simulation of physics. Because we wanted to simulate physics only in 2D we could, code our own implementation of physics, or use one of commercial or open source engines. We chose Box2D[Thorn (2010)], which is open source 2D physiscs engine with posibility to simulate rigid body objects and their collisions.

3.1 World

To create physics simulation we need to create world. World is object that manages memory, objects and simulation. Creation of world is shown in Listing 3:

```
1 self.worldAABB=box2d.b2AABB()
2 self.worldAABB.lowerBound = (-100.0, -100.0)
3 self.worldAABB.upperBound = ( 600.0, 600.0)
4 gravity = (0.0, -10.0)
5
6 doSleep = True
7 self.world = box2d.b2World(self.worldAABB,
      gravity, doSleep)
```

Listing 3. Creation of Box2D world

First we have to create boundaries of the world. We define them as vectors from bottom left (line: 2) to top right (line: 3). Objects have to be inside the boudaries, when an object touch the boundary it gets stuck. Then we define gravity vector (line: 4). The last thing before creation of the world we allow objects to sleep(line: 6). Object that are not moving fall asleep, then are ignored by the engine. Last line of Listing 3 creates the world.

When we have world created we are ready to create objects from outlines.

3.2 Objects

Every object that is simulated in Box2D consists of body and shapes.

Bodies, shapes and collisions Bodies are skelet used by shapes. One body can contain more shapes, but one shape could be attached to only one body. Box2d is rigid body physics engine, that mean that shapes attached to body can not move against other, or body.

3.3 Tesselation

Box2D supports only collisions between convex objects. That is why we need to breake outlines to convex polygons. There is more ways how to break concave objects. We chose Seidel's Triangulation Algorithm implemented by poly2tri Python library

3.4 Implementation

4. FUTURE WORK

Identifikacia objektov Sledovanie objektov a morfing Interakcia hybucich sa objektov zachytenych kamerov s Box2D reprezentaciou

ACKNOWLEDGMENTS

The work was supported by a grant (No. NIL-I-007-d) from Iceland, Liechtenstein and Norway through the EEA Financial Mechanism and the Norwegian Financial Mechanism. This project is also co-financed from the state budget of the Slovak Republic..

REFERENCES

Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library.* O'Reilly, Cambridge, MA.

Thorn, A. (2010). *Game Engine Design and Implementation.* Jones & Bartlett Publishers, Cambridge, MA.