A Framework for Generation of Testsets for Recent Multimedia Workflows

Robert Manthey(✉), Steve Conrad, and Marc Ritter

Department of Computer Science Junior Professorship Media Computing, Technische Universität Chemnitz, StraßE der Nationen 62, 09111 Chemnitz, Germany
{robert.manthey,steve.conrad,marc.ritter}@informatik.tu-chemnitz.de
http://www.tu-chemnitz.de/informatik/mc

Abstract. Our framework offers solution approaches for that inadequacy to be overcome. An abstract description define each test case, its transformation to the designated target platforms as well as the operations and parameters to be processed within the evaluation in such a way that it is independent of any platform. The control of our automated framework workflow is based on Python and Apache Ant which trigger the execution of the described definitions with the result that different tools can be used flexible and purpose-dependent.

We conduct an visual error detection evaluation of FFmpeg, Telestream Episode and Adobe Media Encoder. This consists the creation of single uncompressed images based on the definitions of the test patterns in POV-Ray. After that, they are merged together to video samples which form the platform dependent instances of the test cases. All of these videos are processed with different codecs and encoding qualities during the evaluation. The results are compared with its uncompressed raw material or other test cases.

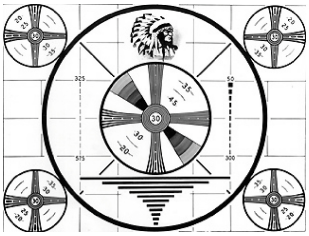
The evaluation shows that the identical test case video file results in visual strongly different outcomes after the encoding. Furthermore some created test cases cause complete losses of the raw information data, ringing artefacts at contrast edges and flicker effects.

Keywords: Framework · Multimedia · Quality analysis · Testing

1 Introduction

Today, a massive amount of video and multimedia data is processed. Cameras observe systems in manufacturing, food production and car traffic. They provide information in autonomous driving cars and advanced driver assistance systems as well as entertainment systems. In a similar way audio and further data are used and sometimes combined to one file or a group of files to make multimedia data. The amount of that data grows as rapid as the their complexity. The resolution increases to HD and more, they get 5.1 to 22.2 surround sound, as well as 3D or 360-degree. The field of application expands form TV and computer screens to huge projectors and small smartwatch like devices. But commonly the

M. Antona and C. Stephanidis (Eds.): UAHCI 2016, Part III, LNCS 9739, pp. 460–467, 2016.
DOI: 10.1007/978-3-319-40238-3_44



(a) RCA Indian-head test image

(b) Lena test image

(c) Frame of the Flower test video

Fig. 1. Commonly used test images and test video

examination of accessibility, correctness, performance and especially quality will be done with old, small size single media samples like Fig. 1a[1], Fig. 1b[2] and Fig. 1c[3] from last century, reaching SD with stereo sound at most.

In principle thereby different steps of a processing chain (Fig. 2) are processed, in order to improve the data, to store them or to show them. Each step has thereby its own characteristics and adds errors, which can be noticed e.g. as picture artefacts shown in Fig. 3. The type of artefacts and their frequency of occurrence are heavily addicted to numerous parameters like the transcoding system, its implementation and settings as well as the input data. Different test patterns exists for different types of image artefacts and due to the innumerable amount of artefacts, they should prompt as many as possible artefact types and make them detectable.

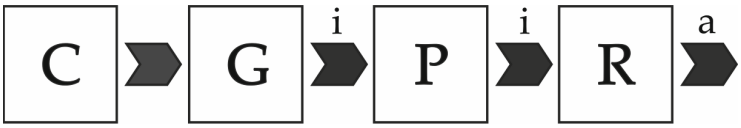
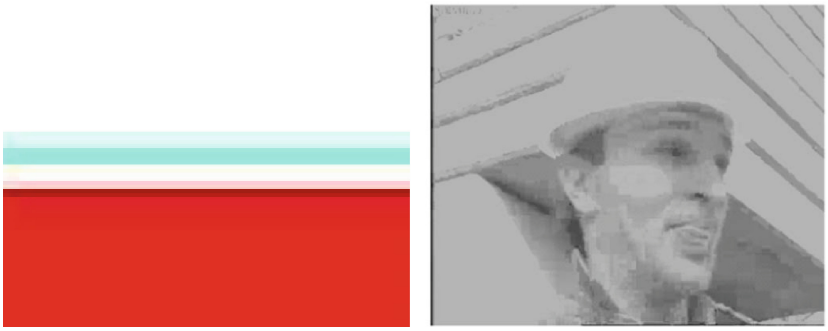


Fig. 2. Processing chain for images "C, input from camera; G, grab image (digitize and store); P, preprocess; R, recognize (i, image data; a, abstract data)."[1]

Many artefacts are nevertheless not detectable, since they will not appear in a single test pattern. They are results of movements, quick image switching or other conditional image transformations which appears in image sequences like videos.

Testsets should not only cause the expected errors, but also make them clearly visible and detectable. In case of single images or nature movies it is e.g. difficult to see slight color differences or single pixel errors. Furthermore in some areas

[1] http://sipi.usc.edu/database/download.php?vol=misc&img=4.2.04.
[2] http://www.forensicgenealogy.info/contest_206_results.html.
[3] http://media.xiph.org/video/derf/y4m/flower_cif.y4m.



(a) Ringing artefact (b) Blocking artefact[4]

Fig. 3. Common artefacts in digital images

like image understanding [3], image retrieval or digital archiving [2] the testsets have to be as compact as possible since otherwise extensive tests would hardly or not at all be possible with such an amount of data. Facing these problems we conceptualized a highly flexible synthetic testset, which was adapted to these specific purposes.

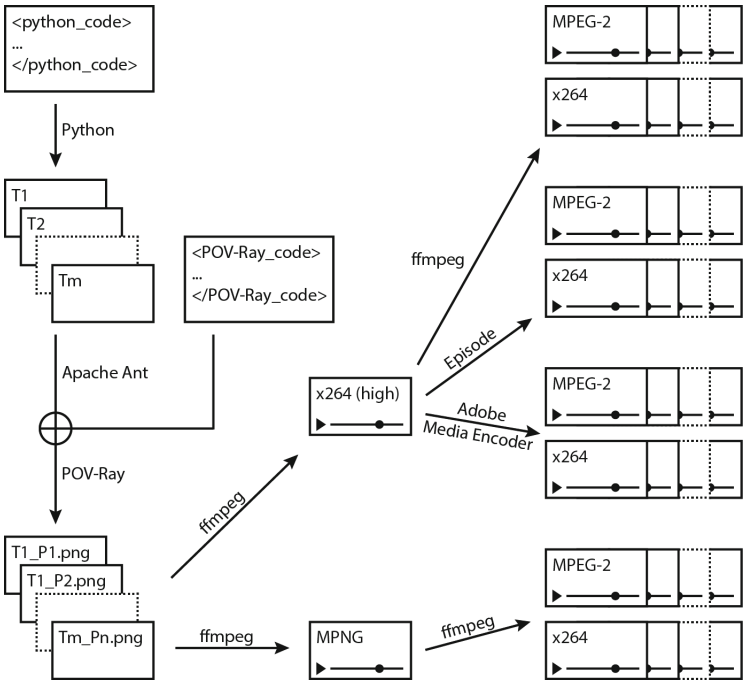


Fig. 4. Schema of the framework to generate the testsets

2 Framework Structure

To generate a synthetic, versatile and flexible testset which is able to detect designated picture artefacts, it is necessary to use a highly adaptable framework from first to last as proposed in Fig. 4. As the fundament we need a vectorized description of the different test patterns like the Scene Description Language used in the open source raytracing software POV-Ray. This abstract scene description is based on parameters and coordinates whereby it is independent from the desired resolution, aspect ratio and file format. Due to this it is easily possible to change the testset or add further test cases in order to adapt the test pattern to changed purposes. In the next step we defined the test cases through the scene description parameters within a Python script which generates the workflow control file. This control file is constructed in such a way that all, a selective amount and even single test cases can be created. It uses Apache Ant to call POV-Ray and to pass the parameters to them. These test cases serve as the input data for the programs to test and as the original material for the comparison with the transcoded results.

3 Testset Generation

We used the description language of the raytracing renderer POV-Ray to define a set of test pattern in a abstract, target and size independent way as shown in Fig. 5a. At the same time a set of descriptions is defined with the Python programming language to form the test sequences (Fig. 5b) as well as the way to handle there execution through the planned program. This forms an Ant-based control[4] file to allow parallel as well as independent execution of each test (Fig. 5c). Further processing steps can be accomplished if planned program needs it to handle the input, as shown in Fig. 4 for cases listed in Table 1.

The created test patterns can be divided in four groups of pattern designs. The first pattern design is a Cartesian grid structure of square blocks, which are provided in edge lengths of 1, 4, 5, 8 and 10 pixels. Except for the 1-pixel-design, the pattern sequences also exist in a rotating version. The images of the second pattern design are composed of 1, 2 or 4 rectangular sections. Additionally the 2-section-sequence are translated perpendicular to their separation line. The 4-section-sequences are also rotating. The third group contains images with stripes in widths of 1, 4, 5, 8 and 10 pixels, which also are rotated and translated in various ways. The last category of test patterns shows a Siemens star in different sizes and with a different number of beams, which are available in a rotating version as well.

Every test pattern sequence exists in four different resolutions, two commonly used and two unusual ones. On the one hand 1920×1080 as a high frequently used resolution for videos and movies as well as for displays and video projectors. On the other hand 1024×768 as an old-standard but still used resolution e.g. in smaller displays, netbooks and mobile devices. Besides we constructed two

[4] https://ant.apache.org/.

```
#while(siemens_beam_count_temp < siemens_beam_count)
polygon (  4,
  <0.0, 0.0, -50>,
  <0, siemens_radius, -50>,
  <sin(1/siemens_beam_count*pi)*siemens_radius,
    sqrt(siemens_radius*siemens_radius -
    (sin(1/siemens_beam_count*pi)*
     siemens_radius*sin(1/siemens_beam_count*pi)*siemens_radius)
    ), -50>,
  <0.0, 0.0, -50>
  texture{
    pigment{ color rgb<0,0,0>}
    finish{ ambient 1.0 diffuse 0.0 }
  }
  rotate<0,0,90-(360*siemens_beam_count_temp/siemens_beam_count)>
  #if (rotation = 1)
    rotate<0,0,abs(360-clock*2)>
  #end
  #declare siemens_beam_count_temp = siemens_beam_count_temp + 1;
}
#end
```

(a) Generic POV-Ray code to generate the siemens star

```
<target name="40007_povray">
    <exec executable="bin\pvengine64.exe">
        <arg line="Output_File_Name=
          siemens_40007_1920_1080_beams16_size3_rot0\
          Output_File_Type=N
          Initial_Frame=0
          Final_Frame=359
          Initial_Clock=0
          Final_Clock=359
          Width=1920
          Height=1080
          Declare=rotation=0
          Declare=siemens_size=3
          Declare=siemens_beam_count=16
          /RENDER siemens.pov /EXIT"/>
    </exec>
</target>
```

(b) Definition of a testcase showing a siemens star without rotation

```
<target name="40007_ffmpeg">
    <exec executable="ffmpeg.exe">
        <arg line="-i  siemens_40007_1920_1080_beams16_size3_rot0\
          40007_mpng.avi
          -vcodec libx264
          -framerate 25
          -r 25
          -y
          -b 400000
          -s 1920x1080
          output_ffmpeg\siemens_40007_1920_1080_beams16_size3_rot0\
          40007_x264_400000.avi"/>
    </exec>
    <exec executable="ffmpeg.exe">
        <arg line="-i  siemens_40007_1920_1080_beams16_size3_rot0\
          40007_mpng.avi
          -vcodec libx264
          -framerate 25
          -r 25
          -y
          -b 1500000
          -s 1920x1080
          output_ffmpeg\siemens_40007_1920_1080_beams16_size3_rot0\
          40007_x264_1500000.avi"/>
    </exec>
```

(c) Definition of the execution of a testcase with 400 kBit/s and 1.5 MBit/s with FFmpeg

Fig. 5. Sample of the control element to generation a testcase with rotation and its execution

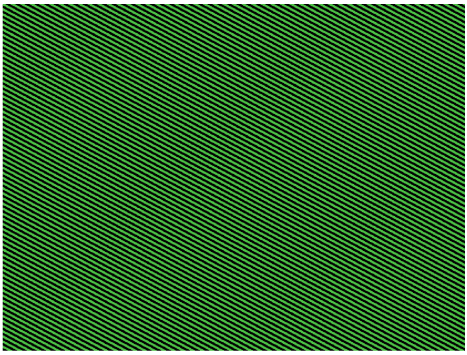
Table 1. Usability of image sequences as direct input data

Encoder	Image sequences usable
FFmpeg	✓
Episode Engine	×
Adobe Media Encoder	×

resolutions out of prime numbers: 1009×631 with an usual aspect ratio of about $16 : 10$ and 997×13 with an uncommon aspect ratio of about $77 : 1$. Moreover each test pattern were generated in five different color sets: A grayscale set, a color set that consists the six complementary colors red, green, blue, cyan, magenta and yellow, a set whereby all $16,777,216$ colors of the RGB color space are randomly changing and two green color sets. These permutations finally

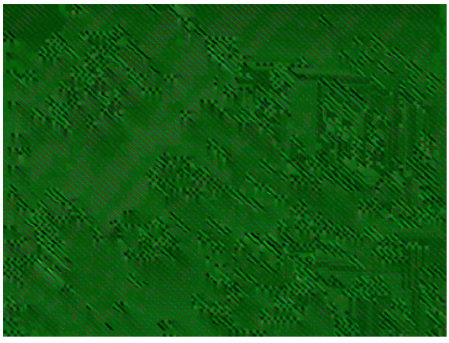
result in over 900 test cases with different structures, transformations, colors and resolutions.

Every test case sequence is made of 360 images of every of these test cases whereby the appropriated transformations and colors change from frame to frame. We used *FFmpeg 2015*, *Telestream Episode 6.4.6* and *Adobe Media Encoder CC 2015.0.1(7.2)* to combine and to transcode the respective 360 frames into different video formats with various qualities. We used the video codecs H.264/x264, which is employed on Blu-ray Discs, in the digital satellite TV broadcast standard DVB-S2 as well as in internet movies and in MP4 files for mobile devices, and MPEG-2, which serves as video format e.g. for DVDs and digital TV broadcast. The test sequences were transcoded to videos with bit rates of 400 kBit/s and 1.5 MBit/s, which are the minimum speed for high quality video calling respectively the recommended speed for HD video calling in Skype, in addition 4.976 MBit/s, 31.668 MBit/s and 83.11 MBit/s, which are the lowest and the highest possible bit rate in DVB-T as well as the highest in



(a) Original strip pattern

(b) Pattern dissolution with spots



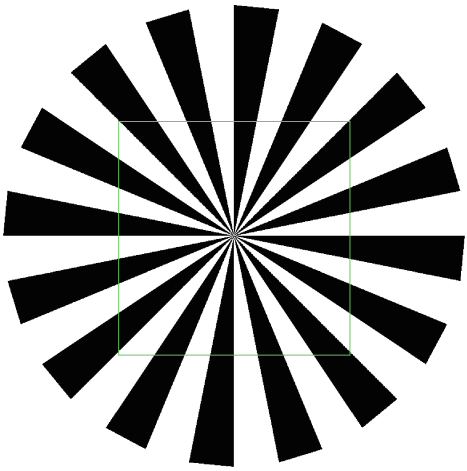
(c) Substantial spots in the pattern

Fig. 6. The strip sample (a) rotating around the picture center leads with FFmpeg, H.264 and 1 MBit/s to the results in (b) and (c). It shows up different unevenly arising spots in varying intensity and with complete dissolution of the original pattern in the first.

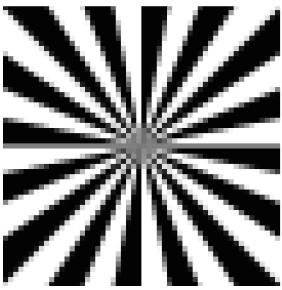
DVB-C, and 15 MBit/s, which is e.g. equal to the MPEG-2 Main Level bit rate. Refresh rate and resolution were not changed during the transcoding process.

4 Experimental Results and Discussion

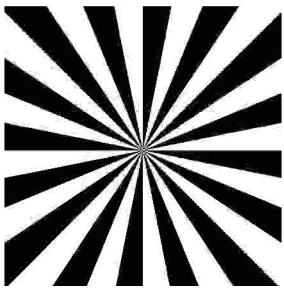
The generated test sequences are processed by the video encoders and empirically examined for remarkable events. The results show that huge single colored structures as well as fence-like vertical or horizontal structures are good encodable. In contrast to that the strip-like content as well as the siemens star pattern



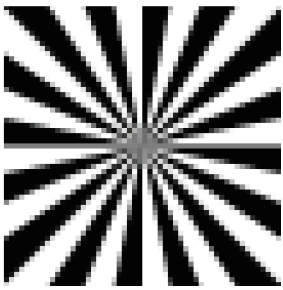
(a) Original siemens star with cutout marking



(b) Central part of the result frame 1



(c) Central part of the result frame 7



(d) Central part of the result frame 15

Fig. 7. The motionless siemens sample (a) leads with Episode Engine, MPEG-2 and 1.5 MBit/s to the results in (b), (c) and (d). After the substantial artefact formation in first frame, a frame-wise quality improvement followed and leads to (c). The next frame (d) show substantial artefacts again and starts a new sequence of improvement with relative good quality conditions in the end.

will create clearly visible artefacts as shown in Figs. 6 and 7. Some are disturbing like in Fig. 6c whereas others impaired the whole image as Fig. 6b.

A further effect of video test sequences is shown in Fig. 7. Still standing similar frames will be modified by the video encoding process and a not existing movement is created.

5 Future Work

In this paper we proposed a new framework for generation of testset for multimedia systems. Since the description of the tests is separated, the applicability of the framework appears very flexible in creating arbitrary testsets and there execution in different environments. We show that the generated testsets can be usable to search for badly influencing effects of performance and quality. We show cases which are too complex to be detected with old-style test images and video sequences.

The next steps incorperate further more complex test patterns, composite sequences to address more artefacts. Test patterns with sound as well as 3D and embeded metadata are to be added. An automatic preliminary investigation of the results could be used to find candidates of problematic testcases. An application to other fields of image processing like robustness research in the field of pedestrian detection can be possible.

Acknowledgments. This work was partially accomplished within the project *localizeIT* (funding code 03IPT608X) funded by the *Federal Ministry of Education and Research* (Bundesministerium für Wissenschaft und Forschung, Germany) in the program of *Entrepreneurial Regions InnoProfile-Transfer*.

References

1. Davies, E.: Machine Vision. Morgan Kaufmann, San Francisco (2005)
2. Manthey, R., Herms, R., Ritter, M., Storz, M., Eibl, M.: A support framework for automated video and multimedia workflows for production and archive. In: Yamamoto, S. (ed.) HCI 2013, Part III. LNCS, vol. 8018, pp. 336–341. Springer, Heidelberg (2013)
3. Ritter, M.: Optimization of algorithms for video analysis: a framework to fit the demands of local television stations. In: Eibl, M. (ed.) Wissenschaftliche Schriftenreihe Dissertationen der Medieninformatik, vol. 3, pp. i-xlii, 1–336. Universitätsverlag der Technischen Universität Chemnitz, Germany (2014). http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-133517
4. Wiegand, T., Sullivan, G.J., Bjntegaard, G., Luthra, A.: Overview of the h.264/avc video coding standard. IEEE Trans. Circuits Syst. Video Technol. **13**(7), 560–576 (2003)