

Перехват и обработка декодированных кадров при помощи OpenCV

- [Описание](#)
- [Сборка OpenCV](#)
- [Реализация перехватчика с использованием OpenCV](#)
- [Тестирование](#)

Описание

Механизм [перехвата и обработки декодированных кадров](#) позволяет интегрировать мощную библиотеку обработки изображений [OpenCV](#). Эта возможность может быть полезна, например, при реализации дополненной реальности на стороне сервера, и в других подобных случаях, и доступна начиная со сборки [WCS5.2.1914](#).

Для того, чтобы реализовать собственный класс для интеграции OpenCV, прежде всего, необходимо собрать саму библиотеку OpenCV.

Сборка OpenCV

Рассмотрим сборку OpenCV 4.9.0 на Centos 7, чтобы обеспечить совместимость с glibc 2.17.

1. Устанавливаем JDK 8 и ANT

```
yum install openjdk-8-jdk ant
```

2. Определяем местоположение каталога установки JDK и настраиваем переменную JAVA_HOME

```
export JAVA_HOME=$(echo $(readlink -f $(which javac)) | sed 's/\bin\/javac//g')
```

3. Устанавливаем CMake 3.6.2 или новее

```
wget https://cmake.org/files/v3.6/cmake-3.6.2.tar.gz
tar -zxf cmake-3.6.2.tar.gz
cd cmake-3.6.2
./bootstrap --prefix=/usr/local
make -j$(nproc)
make install
```

4. Устанавливаем GCC 11 и переключаемся на него для сборки OpenCV

```
yum install centos-release-scl
yum install devtoolset-11-gcc devtoolset-11-gcc-c++
scl enable devtoolset-11 -- bash
```

5. Скачиваем и распаковываем исходные тексты OpenCV

```
wget -O opencv-490.zip https://github.com/opencv/opencv/archive/4.9.0.zip
unzip opencv-490.zip
```

6. Настраиваем сборку

```
cd opencv-490
mkdir build
cd build
cmake ..
```

7. Собираем OpenCV

```
make -j $(nproc)
```

8. Копируем собранные модули в каталог установки WCS

```
cp bin/opencv-490.jar /usr/local/FlashphonerWebCallServer/lib/custom  
cp -r lib/* /usr/local/FlashphonerWebCallServer/lib/so
```

Реализация перехватчика с использованием OpenCV

Для перехвата декодированных кадров необходимо разработать класс на языке Java, реализующий интерфейс `IDecodedFrameInterceptor`. Функция этого класса `frameDecoded()` будет получать декодированные кадры в формате YUV I420, конвертировать их средствами OpenCV в формат RGB, применять размытие, конвертировать результат обратно в YUV I420 и перезаписывать данные фрейма не попиксельно, а одним массивом при помощи метода `Frame.rewriteData()`

TestInterceptor.java

```
// Package name should be strictly defined as com.flashphoner.frameInterceptor
package com.flashphoner.frameInterceptor;

// Import decoded frame interceptor interface
import com.flashphoner.sdk.media.IDecodedFrameInterceptor;
// Import YUV frame description
import com.flashphoner.sdk.media.YUVFrame;
// Import OpenCV classes
import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;

public class TestInterceptor implements IDecodedFrameInterceptor {

    static {
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
    }

    @Override
    public void frameDecoded(String streamName, YUVFrame frame) {

        int width = frame.getWidth();
        int height = frame.getHeight();
        // Calculate chroma height of YUV frame
        int chromaHeight = height * 3 / 2;

        Mat srcYuvMatrix = new Mat(chromaHeight, width, CvType.CV_8UC1, frame.getData());

        // Create RGB matrix for manipulations
        Mat rgbMatrix = new Mat(height, width, CvType.CV_8UC3);
        Mat blurredMatrix = new Mat(height, width, CvType.CV_8UC3);

        // Convert YUV matrix to RGB matrix
        Imgproc.cvtColor(srcYuvMatrix, rgbMatrix, Imgproc.COLOR_YUV2RGB_I420);
        // Apply blur to matrix
        Imgproc.GaussianBlur(rgbMatrix, blurredMatrix, new Size(15, 15), 0);

        Mat dstYuvMatrix = new Mat(chromaHeight, width, CvType.CV_8UC1);
        // Convert RGB back to YUV
        Imgproc.cvtColor(blurredMatrix, dstYuvMatrix, Imgproc.COLOR_RGB2YUV_I420);
        byte[] dstData = new byte[chromaHeight * width];
        // get data from destination matrix
        dstYuvMatrix.get(0,0, dstData);

        // Current method rewrites full frame data with provided dstData
        // This method is recommended for a complete rewrite of the frame, rather than pixel-by-pixel rewriting
        // due to color mismatch
        frame.rewriteData(dstData);
    }

}
```

Затем следует скомпилировать класс в байт-код. Для этого создаем дерево каталогов, соответствующее названию пакета написанного класса

```
mkdir -p com/flashphoner/frameInterceptor
```

и выполняем команду

```
javac -cp /usr/local/FlashphonerWebCallServer/lib/wcs-core.jar:/usr/local/FlashphonerWebCallServer/lib/custom/opencv-490.jar ./com/flashphoner/frameInterceptor/TestInterceptor.java
```

Теперь упакуем скомпилированный код в jar-файл

```
jar -cf testinterceptor.jar ./com/flashphoner/frameInterceptor/TestInterceptor.class
```

и скопируем его в каталог, где размещены собственные Java библиотеки для интеграции с WCS

```
cp testinterceptor.jar /usr/local/FlashphonerWebCallServer/lib/custom
```

Для того, чтобы использовать разработанный класс, необходимо указать имя его пакета в настройках файла [flashphoner.properties](#)

```
decoded_frame_interceptor=com.flashphoner.frameInterceptor.TestInterceptor
```


и перезапустить WCS.

Тестирование

1. Опубликуйте поток в примере Two Way Streaming

Two-way Streaming

Local



test Stop

PUBLISHING


```
{"count": 23}
```

Send payload as object

wss://test1.flashphoner.com:8443 Disconnect

ESTABLISHED

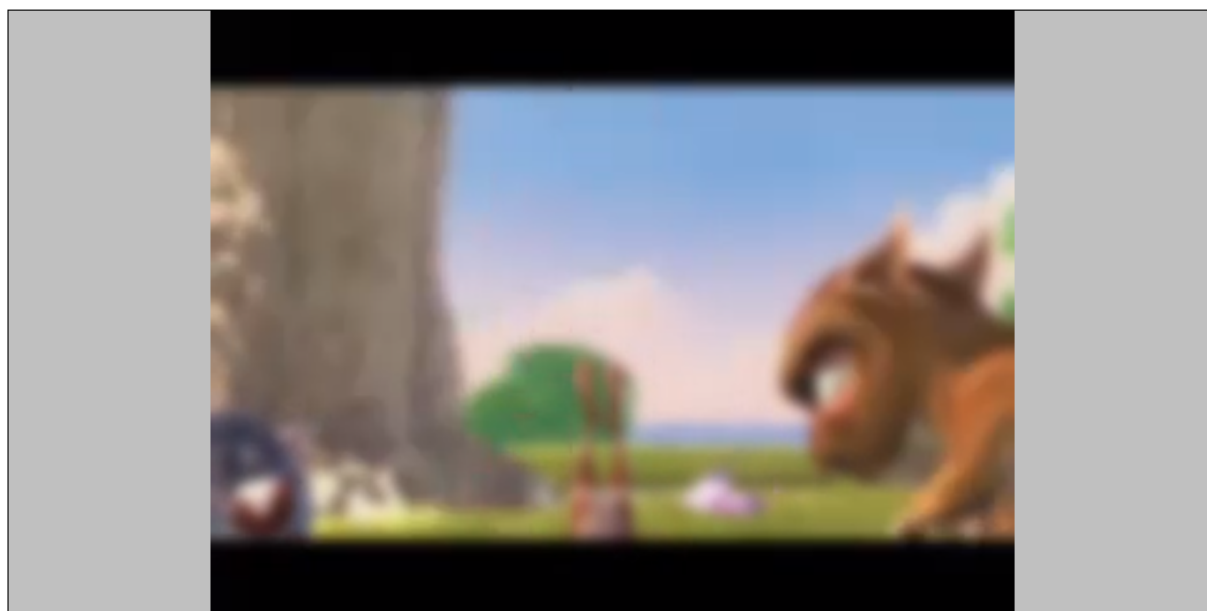
Player



6fe4 Play Available

2. Проиграйте поток в примере Player с указанием разрешения, чтобы включился транскодинг, например <https://test1.flashphoner.com:8444/client2/examples/demo/streaming/player/player.html?resolution=320x240,rgetest1.flashphoner.com> - адрес WCS сервера

Player



WCS URL

wss://test1.flashphoner.com:8443

Stream

test

Volume



Full Screen



PLAYING

Stop

Изображение будет размыто