

## 原 最简单的视频编码器：基于libx264（编码YUV为H.264）

2014年12月23日 00:18:18 阅读数：25974

最简单的视频编码器系列文章列表：

[最简单的视频编码器：编译](#)

[最简单的视频编码器：基于libx264（编码YUV为H.264）](#)

[最简单的视频编码器：基于libx265（编码YUV为H.265）](#)

[最简单的视频编码器：libvpx（编码YUV为VP8）](#)

本文记录一个最简单的基于libx264的H.264视频编码器。此前记录的H.264编码器都是基于FFmpeg调用libx264完成编码的，例如：

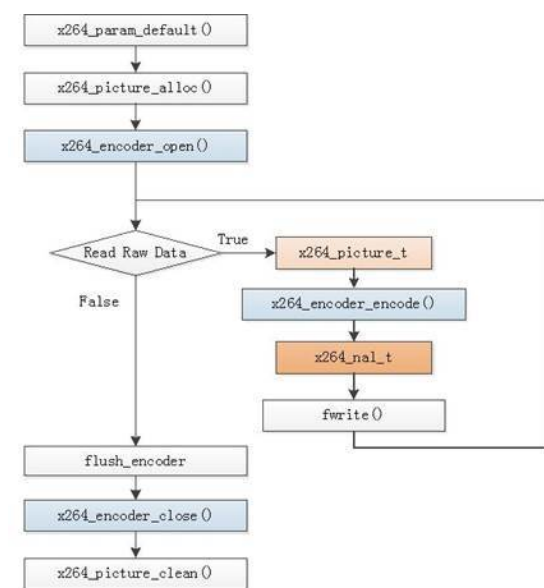
《最简单的基于FFMPEG的视频编码器（YUV编码为H.264）》

相比与上文中的编码器，本文记录的编码器属于“轻量级”的编码器。因为它不再包含FFmpeg的代码，直接调用libx264完成编码。因此项目的体积非常小巧。该编码器可以将输入的YUV数据编码为H.264码流文件。



### 流程图

调用libx264进行视频编码的流程图如下所示。



Simplest Encoder X264  
雷霄骅 (Lei Xiaohua)  
Communication University of China / Digital TV Technology  
Email: leixiaohua1020@126.com  
Website: <http://blog.csdn.net/leixiaohua1020>

流程图中主要的函数如下所示。

x264\_param\_default(): 设置参数集结构体x264\_param\_t的缺省值。

x264\_picture\_alloc(): 为图像结构体x264\_picture\_t分配内存。

x264\_encoder\_open(): 打开编码器。

x264\_encoder\_encode(): 编码一帧图像。

x264\_encoder\_close(): 关闭编码器。

x264\_picture\_clean(): 释放x264\_picture\_alloc()申请的资源。

存储数据的结构体如下所示。

x264\_picture\_t: 存储压缩编码前的像素数据。

x264\_nal\_t: 存储压缩编码后的码流数据。

此外流程图还包括一个“flush\_encoder”模块，该模块使用的函数和编码模块是一样的。唯一的不同在于不再输入视频像素数据。它的作用在于输出编码器中剩余的码流数据。

## 源代码

```

1.  /**
2.   * 最简单的基于X264的视频编码器
3.   * Simplest X264 Encoder
4.   *
5.   * 雷霄骅 Lei Xiaohua
6.   * leixiaohua1020@126.com
7.   * 中国传媒大学/数字电视技术
8.   * Communication University of China / Digital TV Technology
9.   * http://blog.csdn.net/leixiaohua1020
10.  *
11.  * 本程序可以YUV格式的像素数据编码为H.264码流，是最简单的
12.  * 基于libx264的视频编码器
13.  *
14.  * This software encode YUV data to H.264 bitstream.
15.  * It's the simplest encoder example based on libx264.
16.  */
17. #include <stdio.h>
18. #include <stdlib.h>
19.
20. #include "stdint.h"
21.
22. #if defined ( __cplusplus)
23. extern "C"
24. {
25.     #include "x264.h"
26. };
27. #else
28.     #include "x264.h"
29. #endif
30.
31.
32. int main(int argc, char** argv)
33. {
34.
35.     int ret;
36.     int y_size;
37.     int i,j;
38.
39.     //FILE* fp_src = fopen("../cuc_ieschool_640x360_yuv444p.yuv", "rb");
40.     FILE* fp_src = fopen("../cuc_ieschool_640x360_yuv420p.yuv", "rb");
41.
42.     FILE* fp_dst = fopen("cuc_ieschool.h264", "wb");
43.
44.     //Encode 50 frame
45.     //if set 0, encode all frame
46.     int frame_num=50;
47.     int csp=X264_CSP_I420;
48.     int width=640,height=360;
49.
50.     int iNal = 0;
51.     x264_nal_t* pNals = NULL;
52.     x264_t* pHandle = NULL;
53.     x264_picture_t* pPic_in = (x264_picture_t*)malloc(sizeof(x264_picture_t));
54.     x264_picture_t* pPic_out = (x264_picture_t*)malloc(sizeof(x264_picture_t));
55.     x264_param_t* pParam = (x264_param_t*)malloc(sizeof(x264_param_t));
56.
57.     //Check
58.     if(fp_src==NULL||fp_dst==NULL){
59.         printf("Error open files.\n");
60.         return -1;
61.     }
62.
63.     x264_param_default(pParam);
64.     pParam->i_width = width;
65.     pParam->i_height = height;
66.     /*
67.     //Param
68.     pParam->i_log_level = X264_LOG_DEBUG;
69.     pParam->i_threads = X264_SYNC_LOOKAHEAD_AUTO;
70.     pParam->i_frame_total = 0;
71.     pParam->i_keyint_max = 10;
72.     pParam->i_bframe = 5;
73.     pParam->b_open_gop = 0;
74.     pParam->i_bframe_pyramid = 0;
75.     pParam->rc.i_qp_constant=0;
76.     pParam->rc.i_qp_max=0;
77.     pParam->rc.i_qp_min=0;
78.     pParam->i_bframe_adaptive = X264_B_ADAPT_TRELLIS;
79.     pParam->i_fps_den = 1;
80.     pParam->i_fps_num = 25;

```

```

81.     pParam->i_timebase_den = pParam->i_tps_num;
82.     pParam->i_timebase_num = pParam->i_fps_den;
83.     */
84.     pParam->i_csp=csp;
85.     x264_param_apply_profile(pParam, x264_profile_names[5]);
86.
87.     pHandle = x264_encoder_open(pParam);
88.
89.     x264_picture_init(pPic_out);
90.     x264_picture_alloc(pPic_in, csp, pParam->i_width, pParam->i_height);
91.
92.     //ret = x264_encoder_headers(pHandle, &pNals, &iNal);
93.
94.     y_size = pParam->i_width * pParam->i_height;
95.     //detect frame number
96.     if(frame_num==0){
97.         fseek(fp_src,0,SEEK_END);
98.         switch(csp){
99.             case X264_CSP_I444: frame_num=ftell(fp_src)/(y_size*3);break;
100.            case X264_CSP_I420: frame_num=ftell(fp_src)/(y_size*3/2);break;
101.            default: printf("Colorspace Not Support.\n"); return -1;
102.        }
103.        fseek(fp_src,0,SEEK_SET);
104.    }
105.
106.    //Loop to Encode
107.    for( i=0; i<frame_num; i++){
108.        switch(csp){
109.            case X264_CSP_I444:{
110.                fread(pPic_in->img.plane[0], y_size, 1, fp_src); //Y
111.                fread(pPic_in->img.plane[1], y_size, 1, fp_src); //U
112.                fread(pPic_in->img.plane[2], y_size, 1, fp_src); //V
113.                break;}
114.            case X264_CSP_I420:{
115.                fread(pPic_in->img.plane[0], y_size, 1, fp_src); //Y
116.                fread(pPic_in->img.plane[1], y_size/4, 1, fp_src); //U
117.                fread(pPic_in->img.plane[2], y_size/4, 1, fp_src); //V
118.                break;}
119.            default:{
120.                printf("Colorspace Not Support.\n");
121.                return -1;}
122.        }
123.        pPic_in->i_pts = i;
124.
125.        ret = x264_encoder_encode(pHandle, &pNals, &iNal, pPic_in, pPic_out);
126.        if (ret< 0){
127.            printf("Error.\n");
128.            return -1;
129.        }
130.
131.        printf("Succeed encode frame: %5d\n", i);
132.
133.        for ( j = 0; j < iNal; ++j){
134.            fwrite(pNals[j].p_payload, 1, pNals[j].i_payload, fp_dst);
135.        }
136.    }
137.    i=0;
138.    //flush encoder
139.    while(1){
140.        ret = x264_encoder_encode(pHandle, &pNals, &iNal, NULL, pPic_out);
141.        if(ret==0){
142.            break;
143.        }
144.        printf("Flush 1 frame.\n");
145.        for (j = 0; j < iNal; ++j){
146.            fwrite(pNals[j].p_payload, 1, pNals[j].i_payload, fp_dst);
147.        }
148.        i++;
149.    }
150.    x264_picture_clean(pPic_in);
151.    x264_encoder_close(pHandle);
152.    pHandle = NULL;
153.
154.    free(pPic_in);
155.    free(pPic_out);
156.    free(pParam);
157.
158.    fclose(fp_src);
159.    fclose(fp_dst);
160.
161.    return 0;
162. }

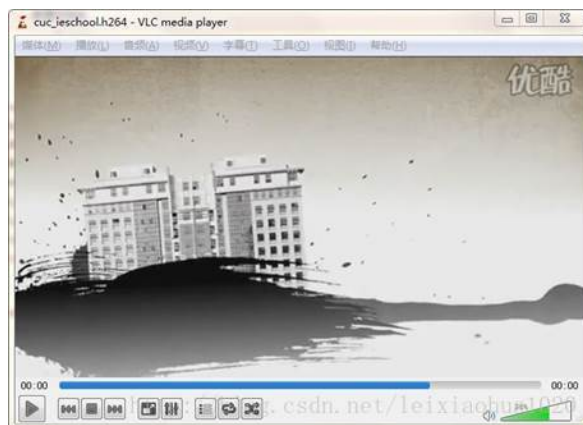
```

## 运行结果

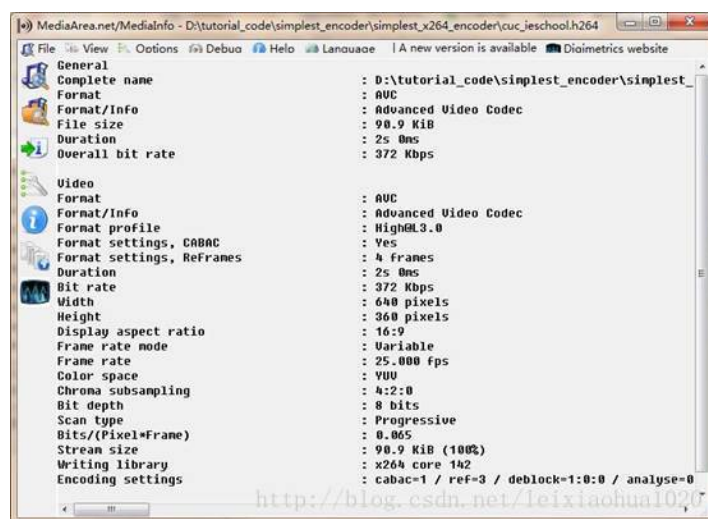
程序的输入为一个YUV文件（已经测试过YUV444P和YUV420P两种格式）。



输出为H.264码流文件。



H.264码流文件的信息如下所示。



## 下载

### Simplest Encoder

#### 项目主页

SourceForge : <https://sourceforge.net/projects/simplestencoder/>

Github : [https://github.com/leixiaohua1020/simplest\\_encoder](https://github.com/leixiaohua1020/simplest_encoder)

开源中国 : [http://git.oschina.net/leixiaohua1020/simplest\\_encoder](http://git.oschina.net/leixiaohua1020/simplest_encoder)

CDSN下载地址 : <http://download.csdn.net/detail/leixiaohua1020/8284105>

该解决方案包含了几个常见的编码器的使用示例：

simplest\_vpx\_encoder：最简单的基于libvpx的视频编码器

simplest\_x264\_encoder：最简单的基于libx264的视频编码器

simplest\_x265\_encoder：最简单的基于libx265的视频编码器

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/leixiaohua1020/article/details/42078645>

文章标签：[x264](#) [h264](#) [视频](#) [编码](#) [YUV](#)

个人分类：[x264](#) [我的开源项目](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com