*Západočeská univerzita*
*Fakulta aplikovaných věd*
*Katedra informatiky a výpočetní techniky*

Semestrální práce z předmětu Multimediální a hypermediální systémy

# Filtr do AviSynthu
# titulky

*Tomáš Chmelař, A02240*
*tomas@tch.wz.cz*

25.4.2005

# Zadání

Mým úkolem je zrealizovat plugin do programu Avisynth. Jeho funkcí je vytvořit závěrečnou sekvenci filmů - posunující se titulky.
Program Avisynth je možné volně stáhnout ze stránek http://www.avisynth.org/
Zde je i přehledný manuál o tom, jak se píší skripty, proto zde tento postup nebudu dále vysvětlovat. Nastíním jenom základní nahrání mého pluginu a jednoduchou ukázku použití.

## *Omezení*

Omezení na vstupní argumenty vytvořeného pluginu jsou následující:

- Podkladový frame a frame s titulky musí mít stejnou šířku.
- Oba vstupní klipy musí být v barevném formátu RGB32

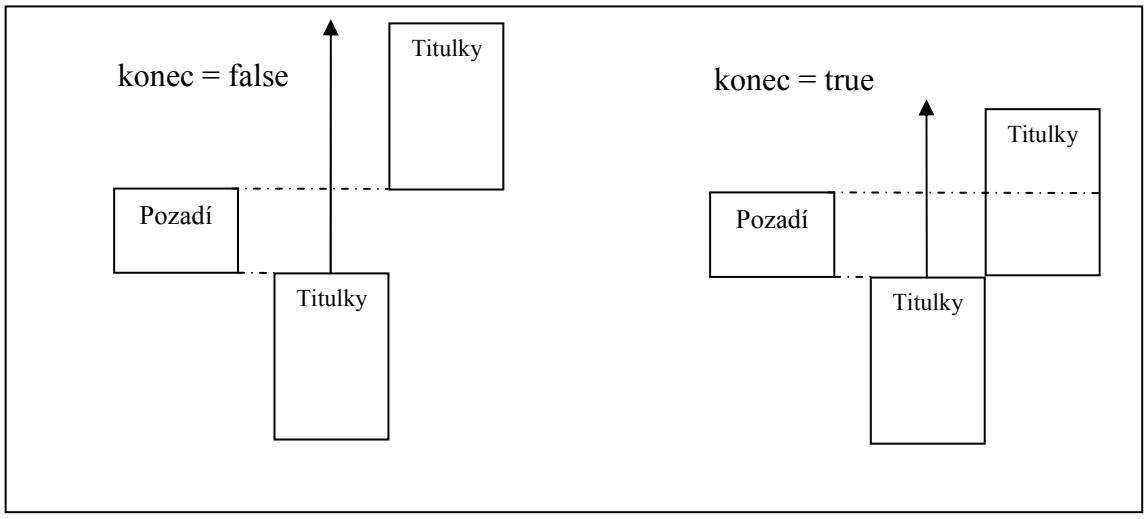
## *Nahrání pluginu*

Nahrání pluginu je realizováno zabudovanou funkcí *LoadPlugin(**string**)*, kde ***string*** udává cestu k DLL souboru s pluginem. Objevuje se na začátku skriptu.

## *Vstupní parametry*

Po zavolání LoadPlugin(titulky.dll) je již přístupná funkce titulky, jejíž vstupní argumenty jsou následující:

1. vstupní klip, který udává celkovou délku výstupního klipu a na který se budou zobrazovat titulky.
2. klip s titulky. Jde o klip, jehož snímek bude představovat titulky, které se budou rovnoměrně posouvat po klipu č. 1
3. proměnná typu boolean, která udává, zda obrázek s titulky se přes vstupní klip přesune celý, nebo zda se posun zastaví, když už byl zobrazen celý snímek:




4. číslo snímku, který se vybere z druhého klipu a jež bude představovat titulky
5. hodnota, jež udává vertikální vzdálenost jednotlivých snímků pro počítání rozmazání (čím větší číslo, tím více je obraz rozmazán)

6. počet snímků před a za momentálním, jež budou použity k rozmazání obrázku
7. x-ová souřadnice pixelu, jehož barva bude v obrázku s titulky označena jako průhledná
8. y-ová souřadnice pixelu, jehož barva bude v obrázku s titulky označena jako průhledná (souřadnice 0,0 je vlevo nahoře)
9. tolerance, s jakou se budou řadit pixely titulků k barvě průhledné:
   - -1 : průhlednost je vyjádřena alfa kanálem obrázku titulků
   - 0 : průhlednost se neuvažuje
   - >0 : čím větší číslo, tím více odstínů se bude považovat za průhlednou barvu

### *Jednoduchý skript pro práci z pluginem:*

```
LoadPlugin("titulky.dll")
# nahraje plugin
titul = ImageReader("tit.tif",0,0)
# do klipu titul vloží do framu 0 obrázek tit.tif
titul=titul.ConvertToRGB32()
# převedeme do barevného prostoru, podporovaného pluginem
poz=BlankClip(length=250,width=375,height=300,pixel_type="rgb32",fps=100,au
dio_rate=48000)
# zde se použije jako podkladový klip černý klip, ale je možné nahrát
#i konkrétní video
return poz.titulky(titul,false,0,0.2,5,5,130)
#výstupem skriptu je výstupní klip z funkce titulky
```

Pokud by uživatel chtěl získat sekvenci titulků z více souborů, stačí vytvořit následující funkci, která jednoduše napojí obrázky za sebe:

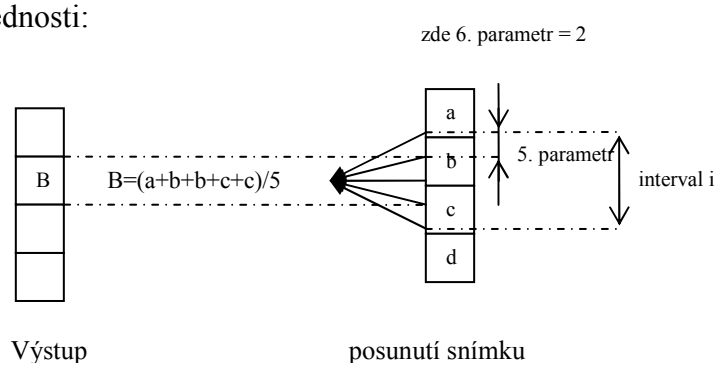
```
Function vice(int start, int end,string soubor) {
return (start<end)?
\StackVertical(ImageSource(soubor,start,start),vice(start+1,end,soubor)):
\ImageSource(soubor,start,start)
}

a zavolat ji:
vice(číslo 1. obrazku, číslo posledního obrázku,"%d.jpg")
```

## Princip řešení

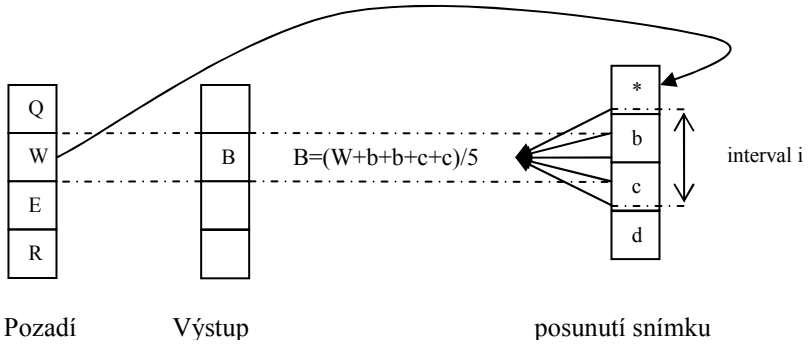
Princip řešení je prostý. Na vstupní klip bude zobrazen jeden snímek z klipu titulků. Posunutí snímku bude v každém framu vypočítáno tak, aby po průchodu celého klipu byl zobrazen celý klip s titulky. Rychlost pohybu titulků je tedy nepřímo úměrná délce vstupního klipu.
V pluginu nebude řešeno načítání obrázku ze souboru, ale vstupní titulky bude představovat frame jednoho ze vstupních klipů, jelikož bude zbytečné implementovat funkce, jež již ve standardním **Avisynthu** fungují. Tím bude docíleno i větší kompatibility v případě, že titulky budou umístěny v sekvenci snímků - klipu.
Aby nedocházelo k trhavému pohybu titulků, bude se hodnota výstupního pixelu v každém bodě počítat jako průměr hodnot barev pixelů vstupního obrázku, pravidelně posunutého ve vertikálním intervalu okolo výstupního pixelu(počet snímků udává 6. parametr). Vzdálenost těchto snímků udává 5. parametr. Pokud bude některý z těchto pixelů označen jako průhledný, bude hodnotou průhledého pixelu hodnota pixelu pozadí ve středu zkoumaného intervalu.

Tímto se zajistí to, že se nebude rozmazávat pozadí v místech, kde je snímek titulků průhledný.

příklad bez průhlednosti:




příklad s průhledností (pixel * je průhledný):




Programově bude toto řešeno tak, že se vypočítají hodnoty posunutí pro první řádek obrázku a pro další řádky se změní ukazatele tak, že se k nim přičte hodnota **pitch** (význam **pitch** je popsán dále).
Průhlednost jednotlivých pixelů se vypočítá při zavolání pluginu.

# Postup řešení:

## *Vytvoření třídy*

Pluginy pro Avisynth se programují v C++ a Výsledný kód se zkompiluje do souboru DLL.

Na začátku každého programu by měl být vložen řádek, jenž importuje soubor avisynth.h.

```
#include "avisynth.h"
```

Jsou v něm uloženy virtuální funkce, jež se obrací na knihovnu Avisynth.dll. Tento soubor (avisynth.h.) je potřeba nakopírovat do adresáře se zdrojovým kódem budoucího pluginu.
Vlastní plugin je jednoduše C++ třída, jenž implementuje rozhraní IClip. IClip obsahuje čtyři základní metody: GetVideoInfo, GetFrame, GetParity a GetAudio. Ve většině případů stačí ale vytvářet filtr, vycházející nikoliv z IClipu, ale z třídy GenericVideoFilter.

```
class titulky: public GenericVideoFilter {
```

Je to jednoduchý, nic nedělající filtr, odvozený s **IClipu.** Výhodou jeho použití je, že ve vytvářeném pluginu se nemusí implementovat metody, které nebudeme potřebovat, jako např. *GetAudio.*

Dále se přidají veřejné metody. V tomto případě jde o filtr s názvem titulky, jež má v Avisynthu 8 argumentů. Pro deklaraci proměnných standardních typů se používá klasické *inr, bool či string*. Deklarace vstupujících klipů se provádí pomocí **PClip**, jež je vlastně takový „chytrý ukazatel" na **IClip.**Obsahuje odkaz na **IClip**, který ale po skončení odstraní z paměti.
Význam jednotlivých vstupních argumentů je popsán výše.
Poslední argument `IScriptEnvironment*` **env** se už ve výsledném pluginu neobjeví a slouží k adresování výstupního klipu pro případ, že chceme vypsat nějaký chybový text. Po převzetí argumentů z **Avisynthu** se jednotlivé hodnoty přiřadí proměnným podle vzorů za dvojtečkou, přičemž první parametr (vstupní klip), převzatý jako **GenericVideoFilter(_child1)** bude přístupný pod ukazatelem **child**. Jde o chráněný člen **GenericVideoFilteru** a je typu **PClip**.

**public:**

```
    titulky (PClip _child1,
          PClip _child2,
          bool _konec,
          int _snimek,
          float _blur,
          int _okolo,
          int _xpointer,
          int _ypointer,
          int _tol,
          IScriptEnvironment* env) :
              GenericVideoFilter(_child1),
              titul(_child2),
              konec(_konec),
              snimek(_snimek),
              okolo(_okolo),
              blur(_blur),
              xtr(_xpointer),
              ytr(_ypointer),
              tol(_tol)
{ init(env);}
```

dále je třeba uvést, jakou metodu z **PVideoFrame** budeme měnit. V tomto případě jde o **GetFrame**. Je to metoda, jež se zavolá pro každý snímek vstupního klipu.

```
PVideoFrame __stdcall GetFrame(int n, IScriptEnvironment* env);
```

Poslední implementovanou metodou bude ta, která se zavolá při spuštění pluginu (inicializaci)

```
void init (IScriptEnvironment* env);
```

## *Nyní můžeme implementovat jednotlivé metody:*

## Inicializace

**void** titulky**::**init **(**IScriptEnvironment***** env**) {**

Jak již bylo napsáno, jedná se o metodu, jež se spustí při zavolání filtru. Prakticky se používá pouze ke kontrole vstupních klipů a parametrů, či k předzpracování vstupů. V našem případě se v této funkci změní průhlednost vstupního snímku s titulky podle vstupních parametrů.

Informace, týkající se vstupního klipu jsou uloženy ve struktuře **vi**. Je typu **VideoInfo** a obsahuje např: velikost obrázků klipu, framerate klipu, formát, audio sample rate a další.
Pokud chceme dostat informace o dalším klipu, jež je zadán jako vstupní parametr, můžeme vytvořit strukturu např. **vi2** a naplnit následovně:

```
vi2=titul->GetVideoInfo();
```

Nyní si naplníme některé proměnné:

```
height = vi.height;
radku = vi2.height;
width = vi.width;
length= vi.num_frames;
framu=vi2.num_frames;
```

Snímek titulků umístíme do struktury **frame2**. Pro jeho získání použijeme interní funkci **Avisynthu - GetFrame**, s parametry: pořadové číslo snímku v klipu (číslováno od 0) a prostředí **env,** jež jsme získali jako vstupní parametr metody:

```
PVideoFrame frame2 = titul->GetFrame(snimek, env);
```

Dále je nutné získat skutečné adresy jednotlivých bodů vstupních a výstupního obrazu. Ty se získají pomocí funkcí **GetWritePtr()** (pro výstupní obraz) a **GetReadPtr()** pro obraz vstupní.

```
const unsigned char* tit = frame2->GetReadPtr();
```

V této fázi tedy máme ukazatel na první bod obrázku v **1. řádce**.
Jednotlivé řádky obrazu jsou uloženy ve stále stejné vzdálenosti od sebe. Hodnotu posuvu mezi řádky získáme pomocí funkce **GetPitch()**

```
const int pitch2 = frame2->GetPitch();
```

U snímku titulků bedeme měnit průhlednost, proto si ho musíme zpřístupnit pro zápis:

```
env->MakeWritable(&frame2);
```

Do proměnné **P2** si uložíme ukazatel na pixely obrázku s titulky.

```
unsigned char* P2=(unsigned char*)tit;
```

Konstanta **tran** bude obsahovat ukazatel na řádku, v níž se nachází pixel, jehož barva bude barvou průhlednou. **YTR** je y-ová souřadnice tohoto bodu.

```
unsigned char* tran=(unsigned char*)(tit+((radku-ytr)*pitch2));
```

Případná chybová hlášení se zobrazují pomocí vstupního argumentu **env** a jeho metody **ThrowError:**

```
env->ThrowError("titulky: clip1 RGB32 color space required");
```

Po zobrazení textu bude běh pluginu ukončen.

Pro testování lze použít řadu podmínek, od standardních **if ( ytr > vi.height )** po speciální **if (!vi.IsRGB32())**

Tímto způsobem tedy můžeme ověřit některé vstupní podmínky:

```
if (!vi.IsRGB32())
   env->ThrowError("titulky: clip1 RGB32 color space required");
if (!vi2.IsRGB32())
   env->ThrowError("titulky: clip2 RGB32 color space required");
if (width!=vi2.width)
   env->ThrowError("titulky: Different width of input clips");
if ( ytr  > height )
   env->ThrowError("titulky: tr yoffset is more then clip tit height");
if ( xtr  > radku )
   env->ThrowError("titulky: tr xoffset is more then clip tit width");
if ( snimek  > framu )
   env->ThrowError("titulky: subtitle number is higher than clip2.lenght");
```

Dalším cyklem projdeme postupně všechny pixely titulků a spočítáme průhlednost jednotlivých pixelů v závislosti vzdálenosti odstínu pixelu od referenční hodnoty a tolerance (vše zadáno jako vstupní parametr):

```
for(int i=0;i<radku;i++)
{
    for(int j=0;j<width;j++)
    {
          if (tol>0)
          {
               long int hh=
                    (tran[xtr*4+0]-P2[j*4+0])*(tran[xtr*4+0]-P2[j*4+0])+
                    (tran[xtr*4+1]-P2[j*4+1])*(tran[xtr*4+1]-P2[j*4+1])+
                    (tran[xtr*4+2]-P2[j*4+2])*(tran[xtr*4+2]-P2[j*4+2]);
               if (hh>(tol*tol))
                    P2[j*4+3]=255;
               else
                    P2[j*4+3]=0;
           }
     }
     P2 += pitch2;
}
```

## Metoda pro zpracování snímků

```
PVideoFrame titulky::GetFrame(int n, IScriptEnvironment* env) {
```

Vstupní parametr **n** nám říká, s jakým snímkem pracujeme.
Nyní je třeba získat vstupní obrázky:
Snímek pozadí umístíme do struktury **frame1**. Pro jeho získání použijeme interní funkci **Avisynthu - GetFrame**, s parametry: pořadové číslo snímku v klipu (číslováno od 0) a prostředí **env,** jež jsme získali jako vstupní parametr metody:

```
PVideoFrame frame1 = child->GetFrame(n, env);
```

Na obrázek s titulky budeme odkazovat pomocí frame2. Snímek je vstupní parametr a udává č. snímku, viz výše:

```
PVideoFrame frame2 = titul->GetFrame(snimek, env);
```

Výstupní obrázek bude odkazován pomocí **dst**. Jelikož jde o obrázek nového klipu, musíme pro něj vyhradit paměť odpovídající velikosti. To se provede pomocí **funkce env->NewVideoFrame(VideoInfo)**, jejíž vstupní parametr plně popisuje vytvářený obrázek výstupního klipu. Jelikož budou parametry výstupního klipu shodné s parametry vstupního, zavoláme funkci s parametrem **vi**:

```
PVideoFrame dst = env->NewVideoFrame(vi);
```

Abychom mohli do nového snímku zasahovat, musíme ho udělat „zapisovatelným“:

```
env->MakeWritable(&dst);
```

***Nyní definujeme některé dále používané proměnné:***

```
int bb,gg,rr;

const unsigned char* PS [20];
float DES [20];
```

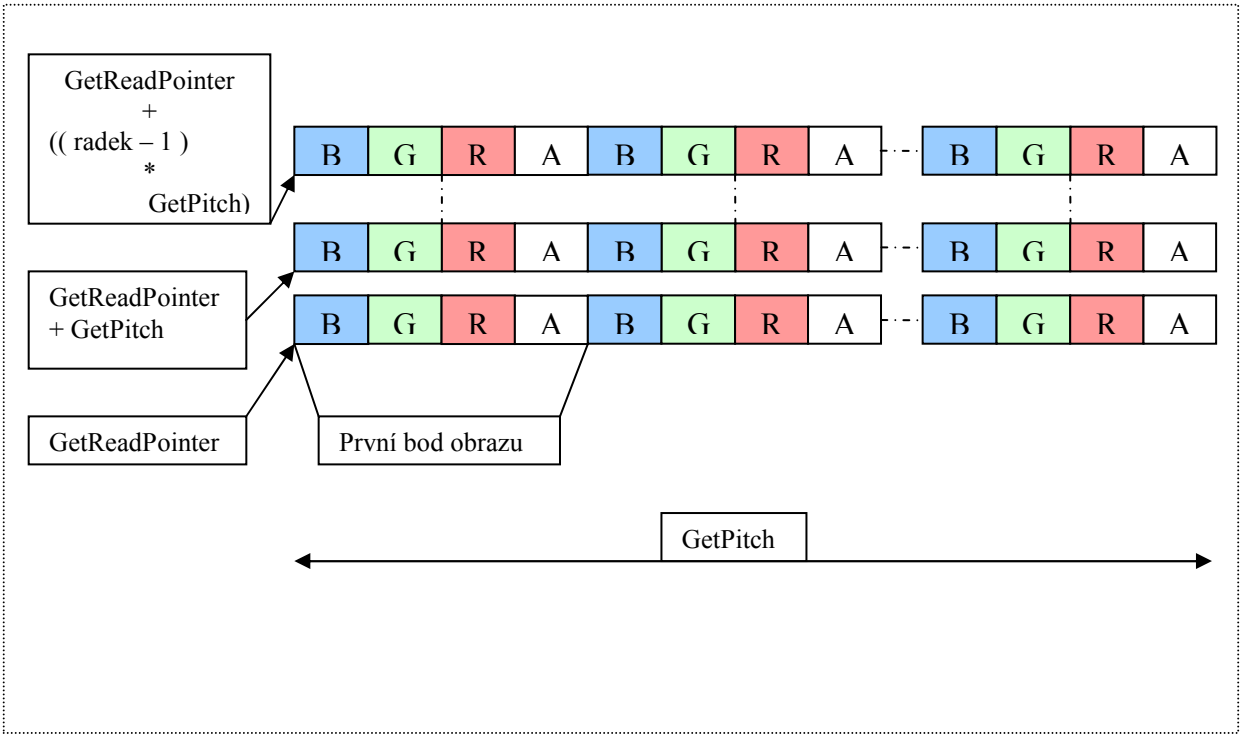
Dále je nutné získat skutečné adresy jednotlivých bodů vstupních a výstupního obrazu. Ty se získají pomocí funkcí **GetWritePtr()** (pro výstupní obraz) a **GetReadPtr()** pro obraz vstupní.

```
unsigned char* p1 = frame1->GetReadPtr();
unsigned char* ven = dst->GetWritePtr();
const unsigned char* tit = frame2->GetReadPtr();
```

V této fázi tedy máme ukazatel na první bod obrázku v **1. řádce**.
Jednotlivé řádky obrazu jsou uloženy ve stále stejné vzdálenosti od sebe. Hodnotu posuvu mezi řádky získáme pomocí funkce **GetPitch()**

```
const int pitch1 = frame1->GetPitch();
const int pitch2 = frame2->GetPitch();
```

Pro formát souborů používáme barevný prostor **RGB32.** První bod se nachází v levém dolním rohu obrázku. Jednotlivé body jsou reprezentovány hodnotami barev **R G B** a průhledností **Alpha.** Schéma uložení těchto informací je následující:




Nyní je potřeba vypočítat krok, o jaký se v každém framu posune snímek s titulky. Jak bylo popsáno výše, je tato hodnota závislá na vstupním parametru **konec**:

```
float krok=(konec)?((float)radku/(length-1)):
                   (((float)radku+height)/(length-1));
```

Následující cyklus vypočte 11 hodnot, rovnoměrně rozložených v intervalu okolo hodnoty **`radku-(n*krok)`**. Tato hodnota udává relativní hodnotu posunutí 1. řádky titulků ve snímku **n**. Vypočtené hodnoty se použijí pro vertikální rozmazání snímků, *viz princip řešení*.

```
for (int i=1;i<6;i++)
{
    DES[i-1]=radku-(n*krok);
    DES[i-1]=DES[i-1]+((0.5*i));
    DES[i+5]=radku-(n*krok);
    DES[i+5]=DES[i+5]-((0.5*i));
}
DES[5]=radku-(n*krok);
```

DAlší cyklus vypočte absolutní hodnoty ukazatelů pro zobrazení první řádky titulků a uloží je do pole **PS**. V **DES** jsou pak desetinné hodnoty posunů, jež by bylo možné dále využít:

```
for (int i=0;i<=(okolo*2);i++)
{
    int pom=(DES[i]);
    PS[i]=tit+(pitch2*(int)pom);
    DES[i]=DES[i]-(int)DES[i];
}
```

Nyní již můžeme začít s naplňováním výstupního **framu:**

```
for (int y = 0; y < height; y++)
{
```

```
    for (int x = 0; x < width; x++)
    {
        bb=0;
        gg=0;
        rr=0;
        for (i=0;i<=(okolo*2);i++)
        {
             if ((PS[i]<tit+(pitch2*radku))&&(PS[i]>tit))  /*

Zde se testuje, zda je PS v rozsahu vstupního obrázku titulků,nebo zda jsme v místě, kde se
zobrazuje jen původní snímek (titulky začínají či končí)
                                                          */
             {
     bb += (int)((PS[i][x*4+0]*PS[i][x*4+3])+ (p1[x*4+0]*(255-PS[i][x*4+3])))/255;
     gg += (int)((PS[i][x*4+1]*PS[i][x*4+3])+ (p1[x*4+1]*(255-PS[i][x*4+3])))/255;
     rr += (int)((PS[i][x*4+2]*PS[i][x*4+3])+ (p1[x*4+2]*(255-PS[i][x*4+3])))/255;
              }
/* na základě průhlednosti počítáme odstín z titulků a podkladu */
              else
              {
                   bb += p1[x*4+0];
                   gg += p1[x*4+1];
                   rr += p1[x*4+2];
/* zobrazujeme původní snímek */
              }
         }
         ven[x*4+0]=bb/((okolo*2)+1);
         ven[x*4+1]=gg/((okolo*2)+1);
         ven[x*4+2]=rr/((okolo*2)+1);
/* závěrečná korekce a uložení */
    }
    for (i=0;i<=(okolo*2);i++)
       PS[i] += pitch2;
    p1 += pitch1;
    ven += pitch1;
/* posun ukazatelů na další řádky */
}
```

Výstupem metody je proměnná typu **PVideoFrame - dst**;

```
    return dst;
```

## Zpřístupnění filtru

Aby bylo možné použít náš nový filtr, potřebujeme funkci skriptovacího jazyka, která vytvoří instanci filtru. To je následující funkce:

```
AVSValue __cdecl Create_titulky(AVSValue args, void* user_data,
IScriptEnvironment* env)
```

***Skriptová funkce napsaná v C++ má 3 argumenty:***

1. args - obsahuje všechny argumenty poskytované do funkce skriptem
2. user_data obsahuje void pointr, který dále poskytneme add funkci (obvykle není potřeba)
3. env obsahuje právě ten IScriptEnvironment pointr, který bude dále předán metodě GetFrame

tělem této funkce může být různé testování vstupních argumentů typu **IsInt()** či jiné.
Výstupem je:

```
    return new titulky(args[0].AsClip(),
                               args[1].AsClip(),
                               args[2].AsBool(),
                               args[3].AsInt(),
                               args[4].AsFloat(),
```

```
                                    args[5].AsInt(),
                                    args[6].AsInt(),
                                    args[7].AsInt(),
                                    args[8].AsInt(),
                                    env );
```

Význam je zřejmý. udává typy proměnných získaných z AviSynthu.
Mohou být následující:

```
AsInt()     -     celočíselná hodnota
AsBool()    -     true/false
AsFloat()   -     desetinné číslo
AsString()  -     řetězec
AsClip()    -     PClip
```

pole AVSValue či nedefinovaný typ.

Poslední funkcí je jediná funkce, která bude z vnějšku přístupná skriptu **Avisynthu** ze souboru **DLL**. Je zavolána skriptovou funkcí **LoadPlugin** v **Avysynthu**:

```
extern "C" __declspec(dllexport) const char* __stdcall
AvisynthPluginInit2(IScriptEnvironment* env) {
```

V ní dáme **Avisynthu** vědět o existenci naší funkce:

```
env->AddFunction("titulky", "ccbifiiii", Create_titulky, 0);
```

***Parametry jsou následující:***

1. jméno naší skriptové funkce
2. řetězec, jež popisuje argumenty :
   - c- klip
   - s- řetězec
   - i- celočíselná hodnota
   - b- true/false hodnota
   - f- desetinné číslo
3. metoda, která se zavolá při zavolání skriptu naší funkce v **Avisynthu**
4. user_data cookie - nechat 0

návratovou hodnotou je textová informace o našem pluginu (autor, popis funkce aj). Je to návratová hodnota skriptu **LOADPLUGIN** a bude většinou ignorována. Můžeme také vrátit 0, jestli chceme.

# Zavěr

Při práci na tomto pluginu jsem se plně seznámil se skriptovacím jazykem programu AviSynth. Je to velice užitečný program, jež produkuje velice dobré výsledky i na počítačích s nižším výkonem. Jeho nespornou výhodou je možnost realtimeového prohlížení vytvářených klipů.
V druhé fázi jsem se naučil programovat pluginy v C++. Jelikož jsem se tomuto jazyku nikdy nevěnoval, bylo to z počátku velice těžké. Postupem času jsem ale zjistil, že pro vytváření jednoduchého pluginu není třeba hlubších znalostí tohoto jazyka a stačí pochopit jeho základní principy.

Celý referát je psán takovou formou, abych kromě popisu kódu mnou vytvořeného pluginu uvedl další obecné poznatky, jež stačí k pochopení základů psaní jednoduchých pluginů těchto typů.