SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1025.

OBRADA I PRIKAZ VIDEO ZAPISA NA TEKSTURAMA OBJEKTA

Krešimir Špes

Zagreb, Veljača 2010.

Sadržaj

UVOD

U većini kompleksnijih grafičkih programa prije ili kasnije se ukaže potreba za prikazom video materijala. Tako na primjer, računalne igre često prikazuju logotipove i međusekvence. Video zapisi se često koriste kao teksturni izvor za 3D objekte, na primjer televizijska emisija koja se prikazuje na 3D modelu TV prijamnika u prostoriji.

Najčešći razlozi za prikaz videa u takvim programima su:

1) Prikaz kompleksne animacije koju bi bilo prekomplicirano izvesti programski, bilo zbog zahtjevnog prikaza ili jednostavno radi uštede vremena programerskom timu.
2) Prikaz animiranog segmenta slike, primjerice vode ili oblaka u pozadini statičnog kadra, animacija likova itd.
3) Ulazni podaci za daljnju obradu, npr. Animairana Visinska karta ili podaci za grafički program za sjenčanje (engl. shader) s kojim se postižu razni efekti.

Primjer korištenja video zapisa kao ulaz programa za sjenčanje je animirana mapa normal vektora [1]. Shader uzima mapu i po njoj računa osvjetljenje 3D objekta te na taj način uz manje poligona postiže se puno veća detaljnost.

Kakva god potreba bila, uvijek je jedan motiv: potreba za uštedom resursa, kako diskovnog prostora, tako i radne memorije.

Najjednostavnije je prikazati video zapis kao slijed tekstura koje se dinamički učitavaju i uništavaju kako video prikaz napreduje. No to iziskuje previše diskovnog prostora i operacija čitanja po mediju na kojem se zapis nalazi.

Stoga, u praksi se animacije niže rezolucije sa manjim brojem slika animiraju direktno iz jedne ili više tekstura, a kompleksnije ili duže animacije putem video dekodera.

Postoje mnogi načini za prikaz videa u današnjim tehnologijama, tako primjerice DirectX SDK pruža programeru mogućnost prikaza videa preko DirectShow [2] filtra na teksturi ili direktno na spremniku (engl. buffer) ekrana.

Postoje komercijalne alternative za koje se mnogi razvojni timovi odlučuju radi jednostavnosti i dobrih performansi, od kojih bi izdvojio najpopularniji: Bink Video[3].

Zajednica koja podžava otvoreni kod (eng. open source zajednica) još uvijek nema kvalitetno rješenje za taj problem što je glavni razlog za ovaj projekt.

VIDEO KODERI / DEKODERI

Video koder je program koji seriju slika komprimira u jednu datoteku. Najpopularniji koderi su tzv. lossy koderi koji koriste karakteristike ljudskog vida kako bi postigli što manju veličinu konačne datoteke uz što manje žrtvovanje kvalitete slike.

Dekoder je program koji iz tako komprimirane datoteke dekodira slike u format pogodan za prikaz.

Obično se uz video informacije, u datoteku kodiraju i dodatne informacije poput zvuka, podnaslova, vremenske oznake poglavlja itd.

Najpopularniji koderi danas su MPEG4, DivX, WMV te Theora.

Za neke od njih postoje i sklopovski ubrzani dekoderi.

THEORA

Theora koder se ističe od ostalih zbog činjenice da je kod javno dostupan (open source) te da nije potrebna licenca za njegovo korištenje u komercijalnim projektima, što su glavni razlozi zašto se mnogi odlučuju upravo za Theoru.

Theora je bazirana na VP3 koderu [4] čiji je kod kompanija On2 Techologies [5] objavila javnosti. Njenim razvojem upravlja fondacija Xiph.Org [6] (Slika 1a) koja razvija i ostale formate otvorenog koda, od kojih su najpopularniji Vorbis i Speex.

Po performansama te omjeru kvalitete i veličine izlazne datoteke, Theoru se uspoređuje sa MPEG4 koderom.

Slika 1: a) Xiph.org foundation b) Ogre3D c) Python-Ogre

OGG / Theora

Xiph.Org je razvio OGG format datoteke koji je primarno dizajniran za reprodukciju audio i video podataka preko interneta, pazeći pritom da se što bolje mogu ispraviti greške pri prijenosu, pogotovo ako neki paket ne stigne na odredište pravovremeno.

Video i audio podaci u datotekama kodiranim Theora koderom su zapakirani u OGG pakete i tako spremljeni kronološki u datoteku.
Program čita pakete sa diska ili preko mrežnog socketa te ih šalje odgovarajućem dekoderu (theora ili vorbis) koji onda, kada pročita dovoljno podataka, izbacuje novu sliku odnosno audio segment.

PLATFORME

Jedan od zahtjeva ovog projekta jest da radi podjednako dobro na svim glavnim PC platformama današnjice, a to su: Windows, MacOS X i Linux.

Pošto libtheora, libvorbis i libogg rade na svim trima platformama te radi standardiziranosti C++ prevodioca, ovaj zadatak nije predstavljao previše problema.

Windows

- Koristi se Microsoft Visual Studio 2008 kompajler
- Kompatibilan sa MinGW kompajlerom (GCC port za Windows)

MacOS X

- Apple-ova verzija GCC kompajlera, koristi se izvršni format koji u sebi sadrži strojni kod za x86 i PowerPC arhitekturu (engl. universal binary).
- Probleme je jedino zadavao PowerPC-ov little endian format.

Linux

- Koristi se GNU-ov GCC kompajler us pomoc autoconf i automake skripti.

Odjeljci koda koje se odnose na određenu plaftormu se uokviruju u #ifdef blokove.

OGRE3D Plugin

Uz ovaj projekt, razvijen je PlugIn za Ogre3D [7] (Slika 1b) koji služi kao sučelje između ova dva sustava.

Dizajniran je tako da automatski prepozna karakteristike sustava na kojemu se izvršava i iskoristi sve hardverske prednosti prilikom procesa dekodiranja.

Dovoljno je u Ogre material skripti navesti ime theora datoteke i plugin automatski na odgovarajuću teksturu zapisuje video slike.

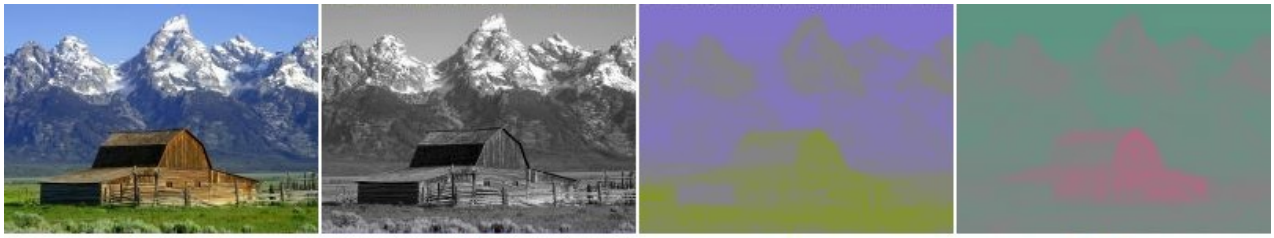
PYTHON INTEGRACIJA

Za Ogre3D plugin napravljeno je sučelje za Python 2.6.x programski jezik.
Taj wrapper projekt je dio većeg projekta pod imenom PythonOgre [8] (Slika 1c).
PythonOgre je python sučelje za Ogre3D koji je postao poprilično popularan u zadnjih par godina.

Sučelje je napravjeno koristeći alate:

- GCCXML [9]
- Py++ [10]
- Boost.Python [11]

Sustav boja YUV



Slika 2: YUV komponente

Theora dekoder izvorno dekodira slike u tzv YUV [12] formatu, točnije Y'CbCr [13]. Razlog leži u tome što je YUV puno pogodniji za kompresiju prilagođenu ljudskom vidu nego RGB [14].

Glavna komponenta Y', zvana “Luma”, je crno bijeli zapis slike.
Ovaj kanal sadrži informaciju o intenzitetu svjetlosti pojedinog slikovnog elementa.

Komponente Cb i Cr se zovu “Chroma” komponente i one utječu na konačnu boju slikovnog elementa.

Pretvorbe između sustava boja *YUV → RGB*

Većina grafičkih programa koristi RGB sustav boja, pogotovo aplikacije koje koriste 3D akceleratorske kartice za prikaz. Stoga je potrebno napraviti konverziju iz YUV u RGB.

Pseudokod za ovu konverziju je:

```
R = 1.164*(Y - 16) + 1.596*(V - 128)

G = 1.164*(Y - 16) - 0.813*(V - 128) - 0.391*(U - 128)

B = 1.164*(Y - 16) + 2.018*(U - 128)
```

što daje rezultate u [0-255] rasponu za svaku komponentu.

OPTIMIZACIJE YUV → RGB

Gore navedeni pseudokod zahtjeva puno kompleksnih operacija sa posmičnim zarezom (eng. floating point operations), i to za svaki slikovni element. Stoga je potrebno uvesti nekakve metode koje bi to pojednostavile i ubrzale.

Tablica

Kako bi se izbjegle operacije s pomičnim zarezom (ili točkom) i nepotrebna množenja, naprave se tablice za svaku komponentu iz gornjih jednadžbi za raspon [0,255].

C++ kod za generiranje tablica:

```
int scale = 1L << 13;
for (int i = 0; i < 256; i++)
{
    temp = i - 128;
    YTable[i]  = (unsigned int)((1.164 * scale + 0.5) * (i - 16));
    RVTable[i] = (unsigned int)((1.596 * scale + 0.5) * temp);
    GUTable[i] = (unsigned int)((0.391 * scale + 0.5) * temp);
    GVTable[i] = (unsigned int)((0.813 * scale + 0.5) * temp);
    BUTable[i] = (unsigned int)((2.018 * scale + 0.5) * temp);
}
```

Vrijednosti u tablicama se pomiču 13 bitova u lijevo kako bi se osigurala dovoljna preciznost korištenjem cijelih brojeva. U sljedećem kodu se te vrijednosti vraćaju 13 bitova udesno.

C++ kod za pretvorbu YUV -> RGB korištenjem tablica:

```
rgbY = YTable[*ySrc];
cu   = *uSrc;
cv   = *vSrc;
rV   = RVTable[cv];
gUV  = GUTable[cu] + GVTable[cv];
bU   = BUTable[cu];
r = CLIP_RGB_COLOR((rgbY + rV ) >> 13);
g = CLIP_RGB_COLOR((rgbY - gUV) >> 13);
b = CLIP_RGB_COLOR((rgbY + bU ) >> 13);
*out=(((r << 8) | g) << 8) | b;
// CLIP_RGB_COLOR dovodi vrijednost u raspon [0,255]
```

Program za sjenčanje

Konverzija YUV → RGB se može optimizirati ukoliko grafičko sklopovlje podržava programe za sjenčanje slikovnih elemenata (engl. pixel shader).
U tom slučaju, slika se u YUV formatu direktno zapisuje na teksturu na RGB komponente te prilikom prikaza na ekranu, obrađuje programom za sjenčanje slikovnih elemenata.

Prilikom korištenja ove metode preporučljivo je:

1) provjeriti podržava li grafičko sklopovlje shadere
2) napisati potreban shader, u GLSL, HLSL, Cg ili grafičkom strojnom kodu
3) napisati dodatne prolaze (npr. ukoliko je potrebno računanje osvjetljenja).

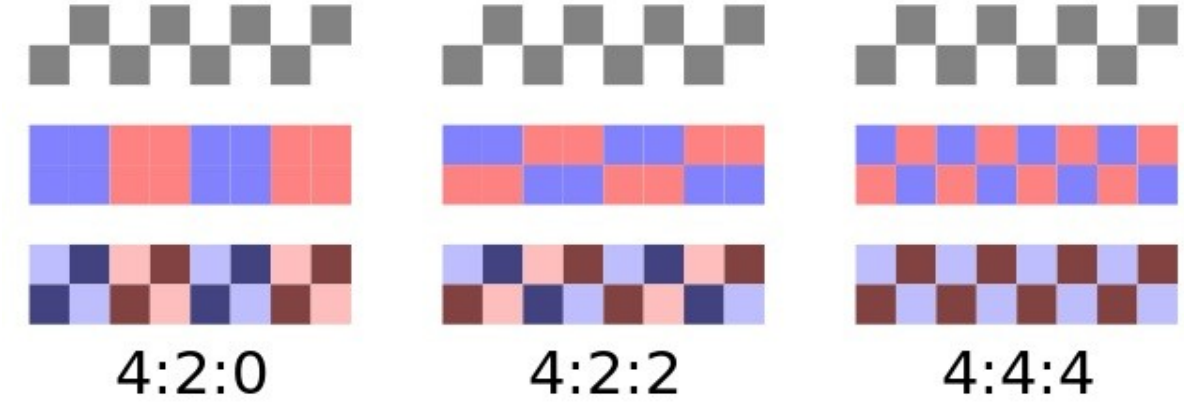
Primjer programa za sjenčanje u grafičkom jeziku GLSL (OpenGL):

```
uniform sampler2D diffuseMap;
void main(void)
{
   vec3 yuv = texture2D(diffuseMap, gl_TexCoord[0].st).xyz;
   float y,u,v,r,g,b;
   y=1.1643*(yuv.x-0.0625);
   u=yuv.y-0.5;
   v=yuv.z-0.5;
   r=y+1.5958*v;
   g=y-0.39173*u-0.81290*v;
   b=y+2.017*u;
   gl_FragColor = vec4(r,g,b,1.0);
}
```

Primjer programa za sjenčanje u grafičkom jeziku HLSL (DirectX):

```
sampler diffuseMap;
float4 ps_main(float2 texCoord : TEXCOORD0) : COLOR0
{
   float3 yuv=tex2D(diffuseMap,texCoord).xyz;
   float y,u,v,r,g,b;
   y=1.1643*(yuv.x-0.0625);
   u=yuv.y-0.5;
   v=yuv.z-0.5;
   r=y+1.5958*v;
   g=y-0.39173*u-0.81290*v;
   b=y+2.017*u;
   return float4(r,g,b,1.0f);
}
```

YUV FORMATI



Slika 3: YUV formati

Praktična stvar kod YUV formata slike i ljudskog vida jest to što Cb i Cr komponente mogu biti niže rezolucije od Y komponente a da pritom slika i dalje dobro izgleda.

Theora podržava sljedeće omjere komponenti prikazanih na Slici 3:

4:4:4
- Chroma i Luma kanali su iste rezolucije

4:2:2
- Horizontalna rezolucija chroma komponenti je dvostruko manja od Luma komponente

4:2:0
- Horizontalna i vertikalna rezolucija chroma komponenti je dvostruko manja od Luka komponente

Najčešći format u Theora video zapisima jest 4:2:0.

Primjer 4:2:0 formata naspram 4:4:4 se može vidjeti na slici 4:

Slika 4: 4:4:4 naspram 4:2:0

PARALELNO DEKODIRANJE

Pošto vrijeme dekodiranja svake slike ovisi o njenoj kompleksnosti, nije poželjno dekodirati slike u istoj dretvi u kojoj se vrti glavni program. Pogotovo u aplikacijama koje kontinuirano osvježavaju prikaz.

Stoga, jedino razumno rješenje jest preseliti dekodiranje u drugu dretvu, dok se glavna dretva bavi samo kopiranjem slikovnih elemenata dekodiranih slika na teksturu ili ekran.

Očito, javlja se potreba za mutexima [15] ili sličnim dretvenim sinkronizacijskim alatima. Ovaj problem postaje izraženiji kada se dekodira više video zapisa odjednom.

Problem je riješen na način da korisnik odbere koliko dekoderskih dretvi želi pokrenuti. Svaka dretva distributerskom sustavu šalje zahtjev za posao i ukoliko posla ima, dretva preuzme obradu jedne od otvorenih datoteka. Pritom mutexom zaključava tu datoteku kako ne bi druge dretve pristupile toj datoteci u isto vrijeme.

Ukoliko program treba dekodirati više video zapisa u isto vrijeme, preporuča se pokrenuti više dretvi, pazeći da pritom broj dretvi ne pređe broj procesora dostupnih operacijskom sustavu.

Nakon što dretva pročita dovoljno OGG paketa da Theora koder može dekodirati sliku, pozove dekodersku funkciju Theore te po potrebi pretvori YUV podatke u RGB.

Dekodirana slika se sprema na stog u priručnu memeoriju tog video objekta i šalje zahtjev distributerskom sustavu za daljnji posao.

Ukoliko jedna dretva obrađuje više dekoderskih poslova, distributerski sustav dekodira datoteke redom jednu po jednu, uzimajući u obzir prioritete (ako ih je korisnik postavio), tako da ukoliko sustav ne može sve stići dekodirati, prioritetniji video zapisi će najmanje slika odbaciti.

Glavna dretva kontinuirano provjerava ima li video objekt novih slika te ukoliko ima, prenosi slikovne elemente na teksturu te uklanja prikazanu sliku sa stoga video objekta.

SIKRONIZACIJA

Najbitnija stvar kod prikaza video zapisa je pravovremeni prikaz slika. Ovom plemenitom cilju postavlja se par težih prepreka. Drugi procesi, performanse sustava, operacijski sutav, pa i sam dekoder mogu prouzročiti kašnjenje u prikazu. Ako je video zapis kodiran u 25 slika/sekundi onda je vremenski prozor za dekodiranje, yuv → rgb pretvorbu, prijenos slike na teksturu te crtanje te teksture na ekran svega 1/25 sekundi!

Program mora mjeriti vrijeme potrebno za svaki korak i ukoliko se uspostavi da slika neće biti dekodirana na vrijeme, mora odbaciti tu sliku i krenuti na sljedeću u interesu sikronizacije.

Moguće je čak napraviti statistička pomagala koji mjere prosječno trajanje svakog koraka te u ranijoj fazi otkriti hoće li doći do odbacivanja slike ili ne.

Priručni spremnik

Kako bi se doskočilo problemu odbacivanja slike, uvodi se pohrana slika u priručnu memoriju (engl. caching). Ovaj proces kontinuirano i unaprijed dekodira slike i zvuk te sprema RGB odnosno YUV rezultate u spremnike priručne memorije.
Kada dođe vrijeme za prikaz, glavna dretva samo uzme prvu dekodiranu sliku sa stoga priručne memorije i kopira slikovne elemente na teksturu.

Dakako, može se dogoditi da se unatoč ovoj optimizaciji ne stigne dekodirati slika na vrijeme, te se pojedini slikovni okviri moraju odbaciti. No ovom metodom, postotak odbačenih slika se svodi na minimum.

Audio/Video sikronizacija

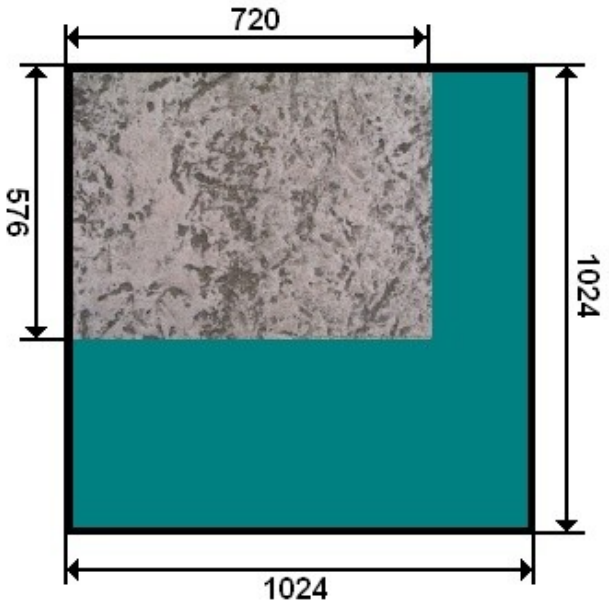
Ukoliko video zapis sadrži samo slike, sinkronizator možemo napraviti kao običan brojač vremena, koji u milisekundama mjeri vrijeme proteklo od početka prikaza.

No, ako imamo i audio zapis u datoteci, moguće je doći do desinkronizacije audia i videa. Stoga, najbolji način za osiguravanje sinkronizacije u tom slučaju jest koristiti sklopovski pokazivač pomaka audio spremnika kojeg reproduciramo kao sinkronizator.

Svaka dekodirana slika u sebi sadrži granularnu poziciju [16] koja se može pretvoriti u redni broj slike, a time možemo (uz poznatu količinu slika po sekundi) i saznati točno vrijeme u kojem se ta slika treba prikazati.

Pa kada brojač prijeđe vrijeme prikaza trenutne slike, skida se nova slika sa stoga i prebacuje na teksturu dok se stara odbacuje, tj. njen memorijski prostor se označava kao slobodnim te stavlja na kraj stoga, čekajući novi sadržaj.

PROBLEM VELIČINE TEKSTURE



Slika 5: Dimenzije teksture naspram dimenzija video zapisa

U programima koje koriste grafičke ubrzivače javlja se problem veličine teksture. Naime, većina grafičkog hardvera radi bržeg crtanja teksturiranih poligona, zahtjeva da dimenzije teksture budu iz brojevnog skupa potencija broja 2.

Stoga, ako imamo video veličine 720x576 kao na slici 5, najbliža veličina teksture koja tome odgovara jest 1024x1024.

Ako sliku jednostavno kopiramo u gornji lijevi kut, programer mora paziti da teksturne koordinate pravilno postavi kako bi ne bi prikazao neiskorišteni prostor teksture.

Neki efekti se ne mogu iskoristiti ukoliko postoje ti neiskorišteni dijelovi, stoga korisnici moraju dimenzije video zapisa prilagoditi njihovim potrebama.

Ukoliko to nije moguće, znači da treba koristiti funkcije za proširivanje slike.

Program korisniku omogućuje zapisivanje izlaznih slika u nativnoj ili proširenoj rezoluciji.
Na primjer, u OpenGL-u možemo koristi funkciju glTexSubImage2D [17] koja osvježava dani segment slike ili glTexImage2D [18] koja prenosi kompletni sadržaj teksture te dimenzije slike koju prenosimo moraju biti dimenzije broja 2.

AUDIO

Audio podaci su sadržani u OGG paketima u datoteci zajedno za Theora OGG paketima. Raspoređeni su paralelno sa Theora paketima radi lakšeg prikaza preko mreže.

Audio koder za Theora zapise jest Vorbis [20], razvijen je od strane iste organizacije koja razvija Theoru i OGG (Xiph foundation)

Audio paketi se šalju Vorbis dekoderu koji na izlazu daje PCM[21] podatke, odnosno amplitude diskretiziranog zvučnog vala u nekoj frekvenciji. Naješće 44 kHz.

Sustav nakon svakog ciklusa dekodiranja nove PCM podatke šalje korisniku koji ih onda reproducira koristeći audio sustav po želji.

Uz ovaj projekt, distribuiran je primjer korištenjem OpenAL [21] sustava, tako da se akumulira N bajtova PCM podataka koji se onda šalju zvučnoj kartici u red za reprodukciju. Nakon što se tih N bajtova reproducira, miču se sa stoga i oslobađa se memorijski prostor u kojem su sadržani.
Koriste se OpenAL-ove funkcije za dohvat vremenskog odmaka kako bi se postigla audio/video sinkronizacija.

Jako je bitno osigurati da sustav uvijek ima dovoljno PCM podataka jer se u protivnom zvuk prekida ako sustav ne stigne dovoljno brzo dekodirati slike. Ljudski dojam je puno osjetljiviji na prekid zvuka nego na odbacivanje slike.
Stoga sustav uvijek čita ogg/vorbis pakete unaprijed te osigurava da audio sustav uvijek ima barem jednu sekundu PCM podataka ispred vremenske pozicije trenutno prikazane slike.

DINAMIČKO POZICIONIRANJE

Ponekad se javlja potreba za dinamičkim pozicioniranjem unutar video datoteke. Najjednostavniji primjer bi bio ponavljanje prikaza video zapisa nakon što se reproducira. Nekad program treba preskočiti N sekundi, pozicionirati se na sljedeće poglavlje itd.

Ključne slike

Pojednostavljeno gledano, video koderi zapisuju slike kao razlike u slikovnim elementima u odnosu na prijašnju sliku.

Radi pozicioniranja, praksa je postaviti svakih N slika tzv. ključnu sliku [22] (eng. key frame). Ključna slika je ništa drugo nego potpuna slika, tj. nije zapisana kao razlika u odnosu na prošlu sliku. Tako da prilikom preciznog pozicioniranja dovoljno je pronaći odgovarajuću ključnu sliku i dekodirati sljedećih N slika kako bi došli do točne slike koja nas zanima.

Pozicioniranje

Ne postoji lagan način za izračunavanje pozicije svake slike unutar komprimirane datoteke posto svaka slika zauzima različiti broj bajtova. Postoji više metoda za ostvarivanje tog cilja:

Seek-mapa

Pri učitavanju datoteke, pročita se cijela datoteka i generira popis svih ključnih slika i njihovoj poziciji unutar datoteke. Tako da je dovoljno pozvati fseek() [23] funkciju za pozicioniranje.

Problem ovog pristupa je što se cijela datoteka mora pročitati što pri velikim datotekama postaje neupotrebljivo.

Tome se može doskočiti tako da program jednom pročita datoteku te rezultate upiše ili u samu datoteku ili u drugu datoteku.

Prva metoda krši Ogg/Theora standard a obje metode zahtijevaju da se datoteka barem jednom pročita. Što nije problem ako korisnik ima fiksan set video materijala koje kani prikazivati.

Binarno pretraživanje

Kompromis koji Xiph.Org preporuča jest koristiti binarno pretraživanje. Tj. binarnom pretragom se pozicionirati, pročitati par OGG paketa i proslijediti ih Theora dekoderu te iz njega izvući apsolutno vrijeme u koje bi se dekodirana slika trebala prikazati i na temelju toga planirati N sljedećih pozicioniranja.

Ova metoda je sporija od seek mape no osigurava maksimalnu kompatibilnost. Što je pogotovo važno ako program mora moći prikazivati korisničke video materijale.

Standard

Nažalost, Ogg/Theora standard ne predviđa seek mapu u zapisu datoteke stoga je najbolje rješenje pri pozicioniranju u Theora zapisu koristiti binarno pretraživanje.
Zanimljiva je činjenica da većina programa za video reprodukciju koji podržavaju Theora dekodiranje obavlja pozioniranje dosta grubo: Otprilike se pozicioniraju na traženu lokaciju te nastavljaju prikazivati. Pošto se najćešće ne pozicionira na ključnu sliku, prvih par slika nakon pozicioniranja izgledaju čudno.

PRIMJENA PROGRAMA

Ovaj program se do današnjeg dana koristio u sljedećim projektima:

1) Legend of Crystal Valley [XY] – Računalna igra hrvatskog studija Cateia Games (Windows i MacOS X)
2) Reviatech-ov interni programski alati za tehničko treniranje [XY]
3) Hotel (radni naziv) – Računalna igra Cateia Games-a
4) Videojumper (radni naziv) – Računalna igra Cateia Games-a koja se igra preko telefona na televizijskom programu
5) Razni projekti koristeći Ogre sučelje (većina nije još objavljena)

Glavni motiv pri izradi ovog sustava jest dati open-source zajednici sustav za prikaz video materijala uz jednostavno sučelje. Tako da se programer ne treba brinuti o problemima prikaza, već samo pozvati par funkcija i prikazati rezultat na svojim teksturama ili direktno na ekranu.

Pretpostavlja se da će sustav biti najčešće korišten za prikaz jednog videa u isto vrijeme, najčešće preko cijelog ekrana kao među sekvence ili logotipovi u računalnim igrama.

Pošto je sustav open-source, razvijat će se dalje po potrebama njihovih korisnika.

Pretpostavlja se da će korisnici s vremenom razviti još sučelja, najvjerojatnije Direct3D [24], Irrlicht [25] sučelje i DirectSound [26] audio sučelje.

OSNOVNO KORIŠTENJE

Program je dizajniran tako da korisniku bude što lakše prikazati video zapis na prikaznom sustavu koji koristi. Idealno, korisnika ne bi smjeli zamarati svi problemi navedeni u ovom radu već treba promatrati sustav kao crnu kutiju.

Primjer: jednostavan program u OpenGL-u (dijelovi OpenGL koda su smanjeni radi preglednosti):

```
TheoraVideoManager* mgr;
Gluint tex_id;

void draw()
{
     glBindTexture(GL_TEXTURE_2D,tex_id);

     TheoraVideoFrame* f=clip->getNextFrame();
     if (f)
     {
          glTexSubImage2D(GL_TEXTURE_2D,0,0,0,clip->getWidth(),
                          f->getHeight(),GL_RGB,GL_UNSIGNED_BYTE,
                          f->getBuffer());
          clip->popFrame();
     }

     drawTexturedQuad(0,0,800,600);

     presentFrame();
}

void update(float time_since_last_frame)
{
    mgr->update(time_since_last_frame);
}

void init()
{
    mgr=new TheoraVideoManager();
    clip=mgr->createVideoClip("primjer.ogg");
    tex_id=createOpenGLTexture(clip->getWidth(),clip->getHeight());
}

void destroy()
{
    delete mgr;
}
```

NAPREDNO KORIŠTENJE

Pored jednostavnog primjera slijede 2 primjera kako se ovaj program može koristiti u atipične svrhe, što nipošto nisu jedine takve primjene.

Prikaz 2D animacija

Animacija likova u dvodimenzionalnom sustavu je uvijek predstavljala problem te se je često moralo pribjeći kompromisnim rješenjima, žrtvujući kvalitetu slike, rezoluciju i/ili količinu slika u sekundi za prikaz animacije.
Još ako pritom morate animirati lika u 8 smjerova što je najčešći slučaj, imate pravu pravcatu noćnu moru!
Uzmimo za primjer lik za računalnu igru koji mora biti visoke rezolucije pošto je predmet fokusa tokom igranja.
Za animaciju hodanja uzmimo rezoluciju 250x300 slikovnih elemenata (RGBA), 20 slika po sekundi i 8 smjerova. Uz laganu upotrebu kalkulatora dolazimo do minimalno potrebnih 46 MB memorije! Samo za animaciju kretanja.

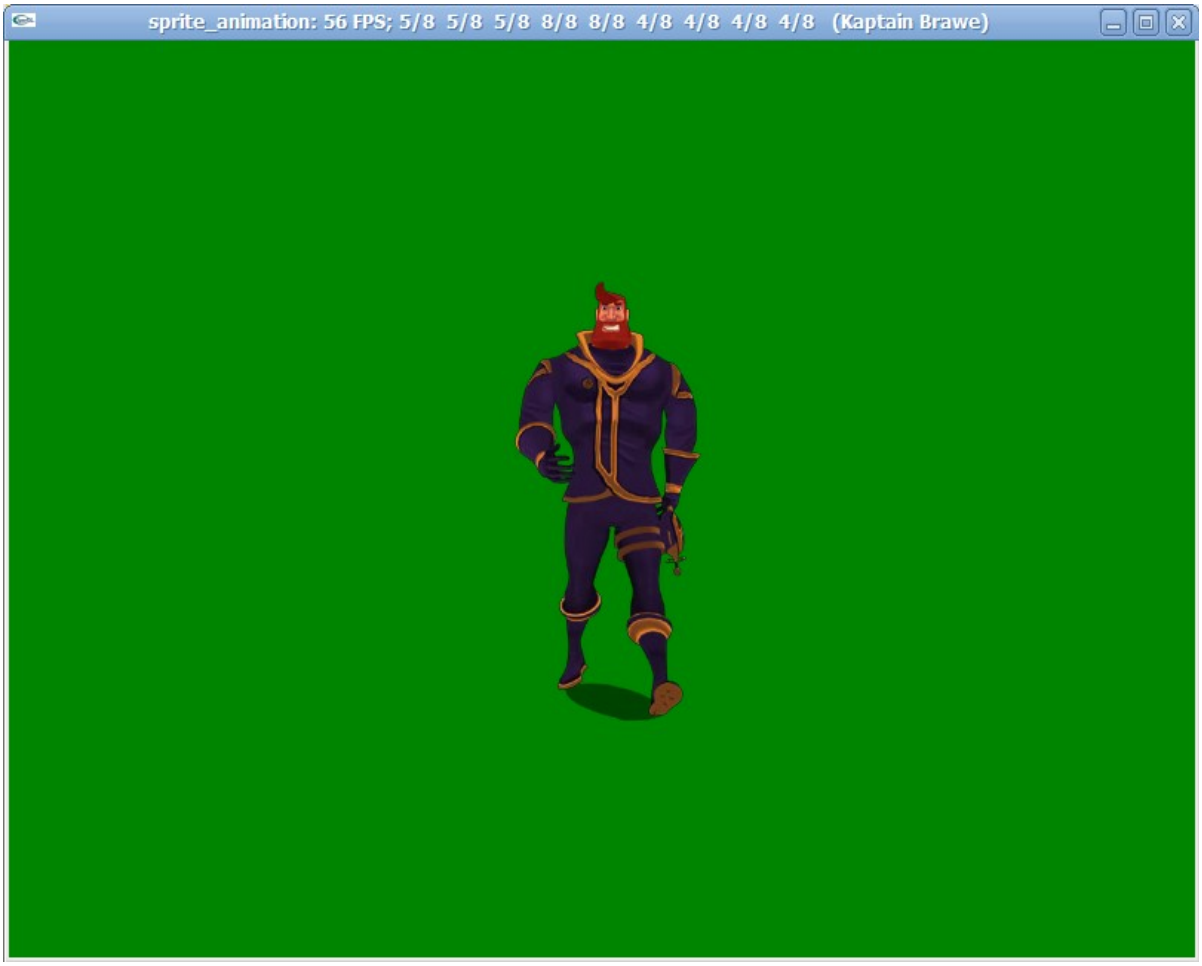
Koristeći ovaj program, uspio sam sniziti memorijske zahtjeve 2D animacije uz malo povećan utrošak procesorske snage.
Napravio sam 8 Theora video datoteka, po jedna za svaki smjer, 20 slika u svakoj, na način da je na lijevoj strani RGB kanal a na desnoj mapa prozirnosti (Slika 6).

Slika 6: Jedna slika spremljena u video datoteku

Sveukupna veličina tih 8 datoteka jest 1.7 MB;
Za glatki prikaz, potrebno je unaprijed dekodirati 2-4 slike, što za 8 smjerova zahtjeva dodatnih 6 MB za pohranu. Rezultat se može vidjeti na slici 7.
Jedini realan nedostatak ove metode jest što zahtjeva konstantan rad procesora za dekodiranje slika. No, taj utrošak je čak i pri ovako velikim slikama relativno mali. Demo program uz prikaz 60 slika po sekundi troši svega 10% procesora.

Koristeći ovu metodu moguće je napraviti jako puno slika uz mali utrošak memorijskog prostora jer su video koderi optimizirani za pokretnu sliku, tj. spremaju se samo oni segmenti slike koji su se promijenili u odnosu na prošlu sliku i zato su te datoteke daleko manje veličine nego kod prve metode.

Pored likova, može se na taj način animirati pozadinska slika (oblaci, voda, trava …), segment slike itd.



Slika 7: 2D lik sa prozirnim segmentima animiran putem video kodera

Prikaz slike u prostoru uz ambijentalno osvjetljenje

Koristeći ovaj program moguće je napraviti razne efekte koji uz minimum truda daju impresivne rezultate, primjer je korištenje video zapisa za prikaz slike na televizoru u 3D prostoru.

Program analizira svaku sliku koju dobije od dekodera tako da izračuna prosječnu boju cijele slike i tu boju postavi kao ambijentalnu boju izvora svjetla televizora.

U ovom primjeru korišten je 3D modelerski alat koji je naprednim tehnikama izračunao osvjetljenje cijele prostorije i to osvjetljenje dodao u teksture objekata na sceni (Slika 8).

Slika 8: Računanje ambijentalnog osvjetljenja iz video slike

ZAKLJUČAK

Na kraju projekta sam poprilično promijenio mišljenje o procesu dekodiranja video zapisa, očekivao sam jednostavan sustav u kojem će program čitati sliku po sliku te prikazivati po potrebi na ekranu dok će se kompletan audio zapis učitati odjednom te reproducirati.
Ubrzo se javila potreba za paralelnim dekodiranjem jer se dosta slika nije stiglo na vrijeme dekodirati te se ujedno i zbog toga broj prikazanih slika po sekundi na grafičkoj kartici srozao na jednoznamenkaste brojeve. Nakon prvih eksperimenata, paralelno dekodiranje se pokazalo kao najboljim rješenjem.

Pretvorba YUV kanala u RGB je predstavljala jedan od većih problema zbog izrazito dugog trajanja te je cijeli postupak trebalo dobro optimirati. Korištenje cjelobrojnih operacija umjesto operacija s pomičnim zarezom i binarno ograničavanje rezultata u rasponu [0,255] je višestruko ubrzalo cijeli postupak.
Ukoliko nije moguće koristiti hardversko sjenčanje slikovnih elemenata, daljnje optimizacije su moguće samo kroz direktno korištenje strojnog koda što bi u korist podržavanja više platformi nastojao izbjeći.

Iznenadila me činjenica da ogg/theora datoteke ne sadrže podatak o dužini trajanja video zapisa u zaglavlje datoteke. Stoga je bilo potrebno pozicionirati se na kraj datoteke i propustiti par OGG paketa kroz theora dekoder kako bi se ta informacija izvukla.
Dodatno me iznenadio nedostatak zapisa o pozicijama ključnih slika. Taj podatak bi znatno olakšao i ubrzao dinamičko pozicioniranje.

Ovaj program je dizajniran tako da ne ovisi o bilo kojem sustavu prikaza (OpenGL, DirectX, SDL...) stoga mu je spektar primjene izrazito širok. Primarno je dizajniran za računalne igre u svrhe prikaza međusekvenci no kao što se iz gore opisanih primjera može vidjeti, primjena mu može biti daleko šira.
Pored međusekvenci, sustav se može koristiti u alatima za transkodiranje video zapisa, za prepoznavanje uzoraka, prikaz online video sadržaja unutar grafičkih ili klasičnih aplikacija, za prikaz animiranih reklama itd.

Audio izlaz je također neovisan o sustavu za reproduckiju. Na korisniku leži na koji način će PCM podatke reproducirati.

Primjena je također moguća na mobilnim uređajima ukoliko proizvođač napravi hardverski dekoder ili barem procesor koji podržava instrukcije za lakše dekodiranje (npr SSE instrukcije [27]).
Većina mobilnih platformi podržava C++ prevodioce i više dretvi što su jedini preduvjeti za korištenje ovog programa. Sklopovka podrška prikaza slike u YUV zapisu bi uvelike ubrzalo prikaz.

Program je otvorenog koda, te je dobro dokumentiran što će nedvojbeno privući buduće programere da pridonesu njegovu usavršavanju i optimiranju što ovom projektu osigurava dobru budućnost i nadam se široku primjenu.

SAŽETAK

Obrada i prikaz video zapisa na teksturama objekta

Video koder je program koji seriju slika komprimira u jednu datoteku. Najpopularniji koderi su tzv. lossy koderi koji koriste karakteristike ljudskog vida kako bi postigli što manju veličinu konačne datoteke uz što manje žrtvovanje kvalitete slike. Theora koder je jedan od takvih te se po kvaliteti uspoređuje za MPEG4 koderom.

Theora koder koristi YUV sustav boja koji je pogodan za komprimiranje slike no javlja se potreba za pretvaranjem u RGB sustav boja kako bi se slika mogla koristiti na teksturi na grafičkom sklopovlju.
Postupak pretvorbe je složen i dugotrajan te je potrebno uvesti razne optimizacije.
Uvedeno je korištenje cjelobrojnih operacija umjesto operacija sa pomičnim zarezom te binarno ograničavanje pretvorenih vrijednosti u rasponu [0,255].
Dodatno je moguće ubrzati pretvorbu koristeći program za sjenčanje slikovnih elemenata no takvo što grafičko sklopovlje mora podržavati.

Pošto Theora koder zapisuje slike kao razlike u odnosu na prošlu sliku, nije moguće predvidjeti vrijeme potrebno za dekodiranje slike što otežava prikaz u glavnoj dretvi te može dovesti do odbacivanja slika.
Stoga je bilo potrebno odvojiti proces dekodiranja od procesa prikaza na način da se dekodiranje preseli na drugu dretvu što je izrazito praktično ukoliko su operacijskom sustavu dostupni više od jednog procesora.

Zbog sikronizacije i glatkog prikaza bilo je potrebo uvesti priručni spremnik slika u koji dekoderska dretva unaprijed dekodira i pohranjuje slike dok glavna dretva po potrebi dohvaća dekodirane slike i prikazuje ih na teksturi.

Ukoliko video datoteka sadrži audio zapis, trenutna pozicija reproduciranog zapisa se može koristiti za sikronizaciju prikaza slika.
U protivnom se koristi obični brojač vremena.

Theora zapis ne sadrži informacije o pozicijama ključnih slika stoga ako korisnik želi preskočiti N sekundi potrebno je pronaći odgovarajuću ključnu sliku korištenjem binarnog pretraživanja.

Program je dizajniran tako da ne ovisi o bilo kojem sustavu prikaza stoga mu je spektar primjene izrazito širok. Dosad se koristio u par komercijalnih projekata te sa svim kvalitetama predviđa se dobra iskoristivost projekta u budućnosti.

Ključne riječi:
theora, ogg, vorbis, glsl, sjenčanje, video koder, YUV, RGB, paralelno dekodiranje

ABSTRACT

Processing and display video on the texture of the object

A video codec is a program that compresses a series of images into one larger file. The most popular codecs are so-called lossy codecs which encode video having in mind the characteristics of human vision to achieve the best possible quality vs. compression ratio. Theora codec is one of them and is often compared to MPEG4 by this ratio.
The Theora codec uses the YUV color system which is best suited for image compression. This however causes problems with display systems that support only RGB, and therefore a conversion method must be introduced.
The YUV → RGB conversion process is complex, time expensive and requires lots of optimizations on a lower level.
In this program, floating point operations in the conversion process have been replaced with integer operations and bitwise operations were used to clamp the converted values to [0,255] range.
It is possible to further optimize this process by using hardware accelerated fragment shaders but not all systems support them.

Since theora codec encodes frames as differences from the previously displayed frame, it is not possible to predict how long it will take to decode a given image which complicates linear frame display in the main thread and causes a lot of dropped frames.
Therefore it was necessary to separate the decoding process from the frame displaying process by separating the decoding to another processing thread. This becomes especially useful if the operating system has access to more then one processor.

Due to synchronization issues and smooth frame display it was necessary to introduce a frame cache buffer that the decoding thread fills by decoding frames in advance while the main display thread fetches frames one by one and displays them.

If the OGG file contains audio data, the current playback location can be used for A/V synchronization. Otherwise a simple timer object will do.

The Theora bit-stream doesn't contain key-frame position information so if the user wants to fast-forward through a video, the system has to find the desired frame by using a bisection search.

The program is designed to be system independent so it can be used regardless of the displaying system used (OpenGL, DirectX, SDL, X11 etc.) and therefore it's potential usage is very promising.
So far it has been used in several commercial project and having all it's features in mind, it is likely it will be used in many more projects.

Keywords:
theora, ogg, vorbis, glsl, shading, video codec, YUV, RGB, parallel decoding

LITERATURA

[1] Normal vektor - http://en.wikipedia.org/wiki/Surface_normal
[2] DirectShow - http://en.wikipedia.org/wiki/DirectShow
[3] Bink Video - http://en.wikipedia.org/wiki/Bink_Video
[4] VP3 Koder - http://www.vp3.com
[5] On2 Technologies - http://www.on2.com
[6] Xiph.org fondacija - http://www.xiph.org
[7] Ogre3D - http://www.ogre3d.org
[8] Python-Ogre - http://www.python-ogre.org/
[9] GCCXML - http://www.gccxml.org/
[10] Py++ - http://www.language-binding.net/pyplusplus/pyplusplus.html
[11] Boost.Python - http://www.boost.org/doc/libs/1_41_0/libs/python/doc/index.html
[12] YUV - http://en.wikipedia.org/wiki/YUV
[13] Y'CbCr - http://en.wikipedia.org/wiki/YCbCr
[14] RGB - http://en.wikipedia.org/wiki/RGB_color_model
[15] Mutex - http://en.wikipedia.org/wiki/Mutual_exclusion
[16] Granularna pozicija - http://www.xiph.org/ogg/doc/ogg-multiplex.html
[17] glTexSubImage2D - http://www.opengl.org/sdk/docs/man/xhtml/glTexSubImage2D.xml
[18] glTexImage2D - http://www.opengl.org/sdk/docs/man/xhtml/glTexImage2D.xml
[19] Vorbis - http://www.vorbis.com
[20] PCM - http://en.wikipedia.org/wiki/Pulse-code_modulation
[21] OpenAL - http://connect.creativelabs.com/openal/default.aspx
[22] Ključna slika - http://en.wikipedia.org/wiki/Key_frame
[23] fseek() - http://www.cplusplus.com/reference/clibrary/cstdio/fseek
[24] Direct3D - http://en.wikipedia.org/wiki/Microsoft_Direct3D
[25] Irrlicht - http://irrlicht.sourceforge.net/
[26] DirectSound - http://en.wikipedia.org/wiki/DirectSound
[27] SSE instrukcije - http://en.wikipedia.org/wiki/Streaming_SIMD_Extensions