OULUN AMMATTIKORKEAKOULU

Janne Luokkanen

ANALOGISET VIDEOSIGNAALIT JA ANALOGISEN VIDEON KAAPPAUS JA GENEROINTI AUDIOLAITTEISTOLLA

ANALOGISET VIDEOSIGNAALIT JA ANALOGISEN VIDEON KAAPPAUS JA GENEROINTI AUDIOLAITTEISTOLLA

Janne Luokkanen
Opinnäytetyö
Kevät 2019
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, laite- ja tuotesuunnittelu

Tekijä: Janne Luokkanen
Opinnäytetyön nimi suomeksi: Analogiset videosignaalit ja analogisen videon kaappaus ja generointi audiolaitteistolla
Opinnäytetyön nimi englanniksi: Analog videosignals and capturing and generating analog video with audio hardware
Työn ohjaaja: Teemu Korpela
Työn valmistumislukukausi ja -vuosi: Kevät 2019
Sivumäärä: 28 + 3 liitettä

Opinnäytetyön aiheena olivat analogiset videosignaalit ja niiden kaappaus ja generointi tietokoneen audiolaitteistolla. Aihe valittiin mielenkiinnosta vanhoja videopelejä ja analogisia videosignaaleja kohtaan.

Tavoitteena oli tutkia analogisia videosignaaleja ja laatia niiden toiminnasta raportti. Tavoitteena oli myös testata, voiko tietokoneen ääniliitännöillä kaapata ja generoida analogisia videosignaaleja.

Työn teoriaosassa selvitettiin, miten kuvaputkitelevisio toimii, miten analogiset videosignaalit rakentuvat, mitä ovat progressiivinen ja lomitettu video, miten lomituksen poisto tapahtuu, mitä ovat analogiset värisignaalit ja miten pikseli toimii. Aiheista piirrettiin myös tarvittaessa kuvaajat. Teoriaosassa selvitettiin myös tietokoneen audiolaitteiston signaalinkäsittelyominaisuuksia.

Toteutusosassa kehitettiin verkkoselaimella toimivat videonkaappaus- ja videongenerointiohjelmat. Videonkaappausohjelma analysoi ja piirtää kuvan valmiiksi tallennetusta videosignaalista. Videon generointiohjelma mahdollistaa oman kuvan piirtämisen ohjelmalla ja pystyy lähettämään analogista videosignaalia reaaliaikaisesti tietokoneen äänilähdön kautta. Kummatkin ohjelmat toteutettiin HTML ja JavaScript verkko-ohjelmointikielillä. Kirjoitetut koodit lisättiin opinnäytetyön loppuun liitteinä.

Työn tuloksena selvisi, että analogisten videosignaalien perustoiminta on melko yksinkertainen, mutta television historian aikana videosignaaleihin on kehitetty monia lisätoimintoja, jotka ovat lisänneet aiheen kompleksisuutta.

Työn toisena tuloksena selvisi, että videosignaalien kaappaus ja generointi äänilaitteistolla on mahdollista, mutta hyvin rajoitettua äänilaitteiston pienen näytteistystaajuuden vuoksi.

Asiasanat: analoginen video, videonkaappaus, komposiittivideo, signaalinkäsittely

SISÄLLYS

SANASTO

NTSC	(National Television System Committee) Enimmäkseen Yhdysvalloissa käytetty televisiojärjestelmä, jossa kuvan päivitystaajuus on 60 Hz (perustuu Yhdysvalloissa käytetyn sähköverkon taajuuteen) (8).
PAL	(Phase Alternating Line) Euroopassa ja muualla maailmassa käytetty televisiojärjestelmä, jossa kuvan päivitysnopeus on 50 Hz (perustuu Euroopassa käytetyn sähköverkon taajuuteen) (8).
Teksti-TV	(engl. teletext) Television digitaalinen toiminto, joka mahdollistaa tekstin ja yksinkertaisen grafiikan lähettämisen televisiolähetyksen mukana. Teksti-TV:tä tukevissa televisioissa on sisäänrakennettu graafinen selain, jolla voi esim. lukea uutisia, etsiä ohjelmatietoja tai lisätä kuvatekstit televisiolähetykseen.

1 JOHDANTO

”Retropelaaminen” ja pelivideoiden tallennus ovat hyvin suosittuja. Monille retropeliharrastajille on tärkeää saada irti paras mahdollinen kuvanlaatu vanhemmista pelikonsoleista, joissa ei ole valmiina digitaalista videolähtöä.

Videopelikonsolien valmistajat ovat tarjonneet omia emulaation perustuvia ratkaisujaan, joissa pelin videokuva voidaan siirtää näytölle suoraan digitaalisessa muodossa modernin videoliitännän kautta. Tällainen on esimerkiksi Nintendon Virtual Console (1).

Erilaiset retropelaamiseen keskittyneet yritykset ovat myös tarjonneet uusia videokaapeleita (2), videosignaalien analogia-digitaalimuuntimia (3) (kuva 1), FPGA-pohjaisia videopelilaitteistoklooneja (kuva 2) ja videopelilaitteistomodifikaatioita, joissa pelikonsoliin lisätään digitaalinen videolähtö (6).

KUVA 1. Open Source Scan Converter (4).

KUVA 2. Analogue Mega Sg (5).

Analogisia videosignaaleja tarkastellessa tulee vastaan sellaisia termejä kuin RGB, YPbPr, CVBS, S-video, Component, Composite, Luma ja Chroma. Työn tavoitteena on oppia analogisten videosignaalien toiminta ja laatia niiden perusteista raportti. Työn toisena tavoitteena on testata, voiko tietokoneesta löytyvillä liitännöillä kaapata ja generoida analogisia videosignaaleja.

Opinnäytetyössä on teoriaosa ja toteutusosa. Teoriaosassa selvitetään analogisten videosignaalien perustoimintaa ja käydään läpi analogisiin videosignaaleihin liittyviä termejä. Kaikki raportin kuvaajat tehdään itse ja suurin osa kirjoitettavasta tekstistä on jo aikaisemmin lukuisista verkko- ja kirjalähteistä opittua tietoa, jota on mahdoton jälkikäteen jäljittää. Toteutusosassa esitellään, miten kaappasin ja analysoin analogista videosignaalia sekä miten toteutin analogisen videosignaalin generoinnin.

Työn aiheen loin omasta mielenkiinnostani analogisia videosignaaleja kohtaan. Pelaan paljon vanhoja konsolipelejä, sillä niissä ei tarvitse huolehtia ohjelmistopäivityksistä tai internetistä ladattavista lisäsisällöistä. Tämä videopeliharrastus on saanut minut kiinnostumaan siitä, miten pelikonsolista television ruudulle siirtyvä kuva luodaan. Lisäksi minusta on mielenkiintoista laajentaa tietokoneessa jo olemassa olevien liitäntöjen käyttömahdollisuuksia muihin tehtäviin kuin ne olivat alun perin suunniteltu.

Tässä opinnäytetyössä tulen keskittymään nimenomaan analogisiin videosignaaleihin. Nykyään yleisesti käytetyt videoliitännät, kuten HDMI ja DisplayPort, ovat digitaalisia, joissa käytettävien korkearesoluutioisten signaalien kaappaus on tämän opinnäytetyön laajuuden ulkopuolella.

2 ANALOGISTEN VIDEOSIGNAALIEN TOIMINTA

Ennen televisiota oli filmikamera, jossa videota nauhoitettiin kaappaamalla kokonaisia kuvia peräkkäin filminauhalle ja näitä kuvia voitiin sitten filmin kehittämisen jälkeen toistaa filmiprojektorilla. Filmikuvaa ei ollut mahdollista lähettää elektronisesti radiotien kautta, joten uusi tapa kaapata ja toistaa videota oli keksittävä. Elektronista kuvan siirtoa varten piti keksiä keino muuttaa kuva sähkösignaaliksi ja sähkösignaali takaisin kuvaksi. Useat keksijät kehittivät omia keinojaan tämän toteuttamiseksi ja näistä lopulta onnistunein oli katodisädeputkitelevisio. (7.)

2.1 Kuvaputkitelevisio

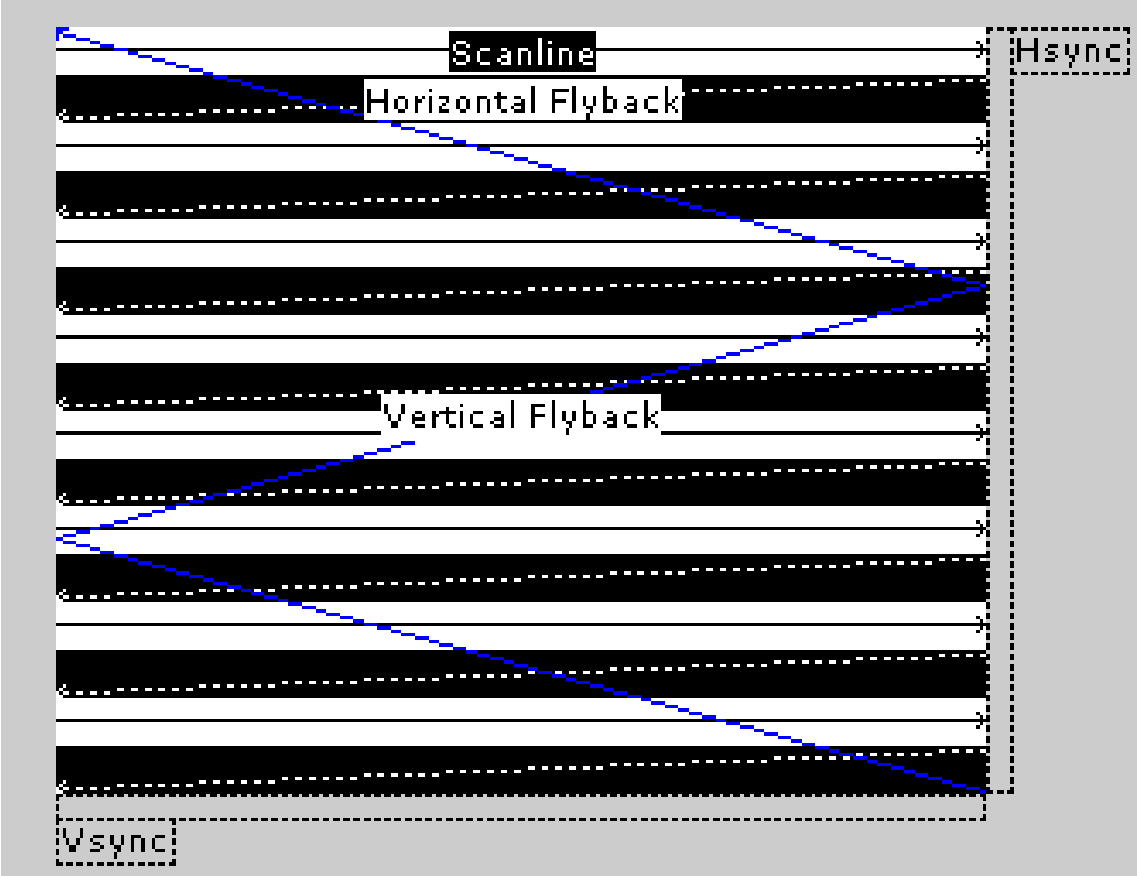
Katodisädeputkitelevision (Cathode Ray Tube, CRT) toiminta perustuu elektronisäteeseen, jota ammutaan fosforiloisteaineella peitettyyn lasipintaan tyhjiöputken sisällä. Tämä lasipinta toimii television näyttöalueena. Elektronitykki (engl. electron gun) luo hohtavan pisteen loisteaineeseen ja pistettä liikutetaan hyvin nopeasti koko näytön alueella, valaisten sen. Koko näyttöalue pitää päivittää jatkuvasti, koska loisteaine himmenee nopeasti. Loisteaineen jatkuva himmeneminen ilmenee kuvan välkkymisenä. Elektronisäde liikkuu niin nopeaa, että koko television kuva-alue piirretään joko 50 (PAL) tai 60 (NTSC) kertaa sekunnissa luoden illuusion vakaasta ja jatkuvasta kuvasta. Analogisen videosignaalin jännitetasoa muuttamalla voidaan ohjata elektronitykin luoman pisteen kirkkautta. Elektronitykin maksimijännitteessä piste on valkoinen ja nollassa piste on musta. Tämän pisteen kirkkaussäädön tarkka ajoitus mahdollistaa harmaasävyisen kuvan muodostamisen television ruudulle.

Elektronisädettä liikutetaan ruudulla juovittain (engl. scanline), niin että jokainen juova alkaa ruudun vasemmasta reunasta ja päättyy ruudun oikeaan reunaan. Televisioruutu on jaettu yleensä joko 625 (PAL) tai 525 (NTSC) vaakajuovaan. Näistä juovista tosin vain 576 (PAL) tai 480 (NTSC) näkyy ruudulla. Lisäksi jokaisen juovan alkuosa ja loppuosa on asetettu mustaksi. Nämä juovien ominaisuudet liittyvät seuraaviin tekijöihin: Hsync, Vsync ja overscan.

2.2 Hsync- ja Vsync-pulssit

Televisio liikuttaa elektronitykin sädettä koko ruudun alueella (kuva 3). Säteen liikuttaminen aloitetaan ruudun vasemmasta ylänurkasta ja sädettä liikutetaan toistuvasti vasemmalta oikealle ja ylhäältä alaspäin. Jotta kuva pystyttäisiin piirtämään oikein ja ilman vääristymistä, on videosignaaliin lisättävä tieto, jolla säteen liike voidaan tahdistaa oikein.

Tätä varten analogisissa videosignaaleissa on Hsync- ja Vsync-pulssit, joita kutsutaan myös nimellä Horizontal ja Vertical blanking interval. Hsync määrittää tahdin, jolla elektronisäde siirtyy juovalta toiselle, eli sen, montako juovaa piirretään sekunnissa (Hfreq). Vsync määrittää tahdin, jolla elektronisäde siirtyy ruudun pohjalta takaisin ruudun vasempaan ylänurkkaan, eli sen, montako kuvaa piirretään sekunnissa (Vfreq). Sync-pulssien aikana elektronisäde on täysin sammutettu, jotta ruudulle ei piirrettäisi mitään säteen paluuliikkeen (engl. flyback) aikana.



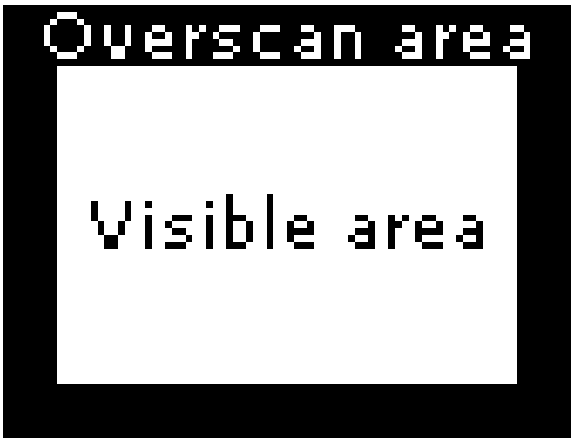
KUVA 3. Elektronitykin liikerata, jonka videosignaali määrittää.

Jokaisen juovan lopussa on lyhyt, muutaman mikrosekunnin pituinen pulssi, jossa videosignaalin jännitetaso lasketaan joko nollaan tai negatiiviseksi (mustan värin jännitetason alle). Tätä pulssia kutsutaan nimellä Hsync (Horizontal Sync), jonka aikana elektronisäde siirretään takaisin ruudun vasempaan reunaan ja seuraavan juovan alkuun. Hsync pulssin pituus riippuu juovien piirtotahdista (Hfreq).

Kun elektronisäde on päässyt viimeisen näkyvän juovan loppuun eli kuvaruudun pohjareunaan, videosignaalin jännitetaso laitetaan negatiiviseksi/nollaan noin millisekunniksi, jolloin säde palaa takaisin kuvaruudun vasempaan ylänurkkaan. Tätä pitkää pulssia kutsutaan nimellä Vsync (Vertical Sync). Vsync-pulssin pituus riippuu kuvan piirtotahdista (Vfreq). Digitekniikan kehittyessä Vsync-pulssia oli myös mahdollista hyödyntää television uusiin toimintoihin upottamalla sinne esim. teksti-TV-dataa, joka ilmenee Vsync-pulssissa binaarisina jännitemuutoksina (tai mustavalkoisina raitoina).

2.3 Overscan

Useimmat kuvaputkinäytöt eivät pysty piirtämään videosignaalin kuva-aluetta kokonaan, vaan osa kuvasta jää piiloon ruudun ulkopuolelle. Tästä syystä monesti videosignaaliin jätetään tarkoituksella tyhjää tilaa kuva-alueen reunoille (kuva 4). Televisiokameran näköalue pitää sommitella huolella, jotta kaikki mitä halutaan näkyvän kuvassa, sijoittuisi kuva-alueen näkyvälle osalle ja overscan-alueelle ei jäisi mitään ylimääräistä näkyviin. Myöhemmin overscan-aluetta alettiin täyttämään mustalla värillä. Overscan-alueelle voidaan myös upottaa digitaalista dataa.



KUVA 4. Videosignaalin näkyvä alue ja overscan-alue.

2.4 Gammakorjaus

Kuvaputkitelevision elektronitykin säteen kirkkauden säätö ei toimi lineaarisesti vaan eksponentiaalisesti. Lineaarinen videosignaali pitää gammakorjata kuvaputkelle sopivalla gammakertoimella. NTSC-televisiojärjestelmä käyttää gammakerrointa 2,22 ja PAL-televisiojärjestelmä gammakerrointa 2,8. Alun perin televisiokamerat käyttivät valon kaappaamiseen tyhjiöputkisensoreita, joiden tuottama videosignaali on lähes suoraan käänteisgamma kuvaputkitelevision tyhjiöputken elektronitykille. Tästä syystä televisiokameroiden tuottama signaali oli jo valmiiksi gammakorjattu ja voitiin käyttää helposti kuvaputkitelevisioiden signaalina. Myöhemmin tämä johti kuitenkin ongelmiin, kun videokuvaan haluttiin lisätä grafiikkaa, sillä gammakorjattuun kuvaan ei voi lisätä toista kuvaa lineaarisesti. Tätä varten piti kehittää videomiksereitä, jotka pystyivät käsittelemään epälineaarisia signaaleja. (8.)

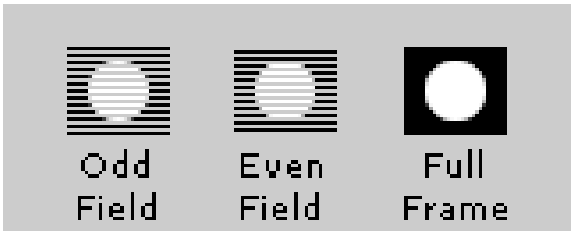
2.5 Progressiivinen ja lomitettu video

Kuten aikaisemmin mainittiin, televisioruutu on siis jaettu joko 625 (PAL) tai 525 (NTSC) vaakajuovaan. Nämä juovat muodostavat kehyksen (engl. frame), joka vastaa yhtä kokonaista kuvaa.

Analogisten televisiolähetysten radiokaistanleveyden rajoitteiden vuoksi kaikkia juovia ei voitu lähettää 50/60 kertaa sekunnin aikana. Ongelma olisi voitu ratkaista laskemalla kuvan päivitysnopeus puoleen (25/30), mutta tämä olisi hidastanut elektronisäteen liikenopeutta niin paljon, että televisiokuva olisi välkkynyt häiritsevästi. Tästä syystä kehykset päätettiin jakaa parillisiin ja parittomiin vaakajuoviin, jotka lähetettäisiin vuorotellen. Tällöin vain puolet kuvan vaakajuovista piirretään ruudulle kerrallaan, mutta kuva piirretään edelleen 50/60 kertaa sekunnissa. Tätä vain puolet kehyksen vaakajuovista sisältävää kuvaa kutsutaan nimellä kenttä (engl. field).

Videota, jossa kehykset eli kaikki kuvan vaakajuovat piirretään 50/60 kertaa sekunnissa, kutsutaan progressiiviseksi videoksi (engl. progressive video) ja videota, jossa kentät eli puolet kuvan vaakajuovista piirretään 50/60 kertaa sekunnissa, kutsutaan lomitetuksi videoksi (engl. interlaced video) (kuva 5). Vsync-

pulssiin on lisätty tieto siitä, ovatko seuraavan kentän vaakajuovat parillisia vai parittomia, jos kyseessä on lomitettu videosignaali.

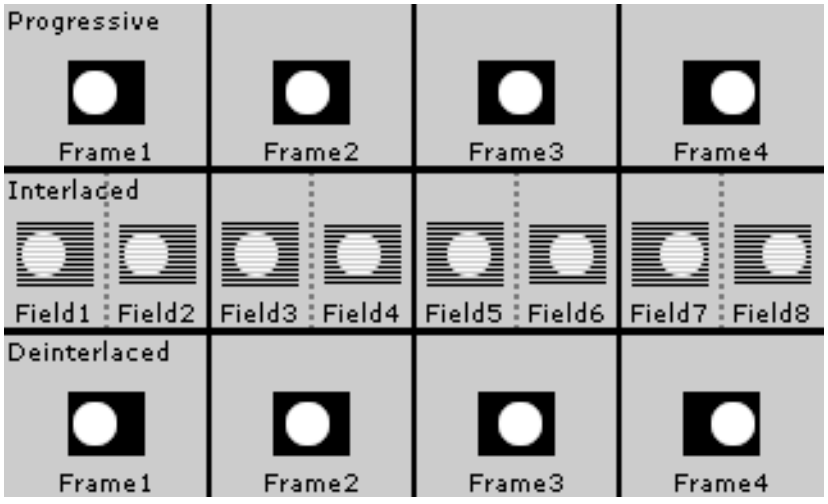


KUVA 5. Lomitetun videon kentät ja progressiivisen videon kehys.

2.6 Lomituksen poisto

Lomituksen poisto (engl. deinterlacing) mahdollistaa lomitetun videon muuntamisen progressiiviseksi videoksi. Lomituksen poisto vaaditaan, mikäli videosignaalia käsittelevä tai toistava laite tukee vain progressiivista videota.

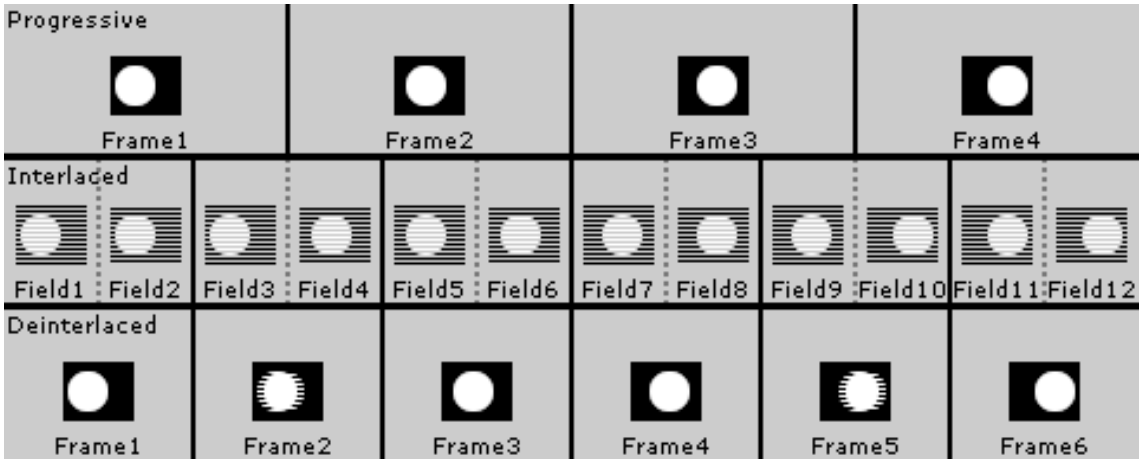
Jos alkuperäisen videokuvan päivitysnopeus on täsmälleen puolet lomitetun videon päivitysnopeudesta, eli alkuperäinen progressiivinen video päivittyy esimerkiksi 30 kertaa sekunnissa ja lomitettu video päivittyy 60 kertaa sekunnissa, ovat alkuperäisen progressiivisen videon kehykset täysin palautettavissa (kuva 6).



KUVA 6. Lomituksen poisto, jossa alkuperäinen video 30 Hz.

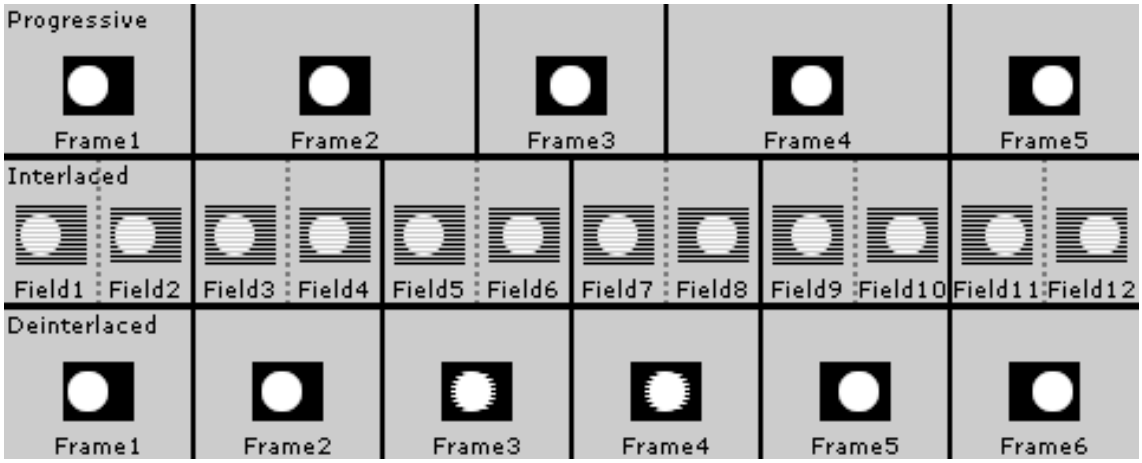
Tämä on mahdollista, koska vuoroittain vierekkäiset parilliset ja parittomat kentät kuuluvat samaan aikahetkeen ja muodostavat yhdessä kokonaisen kehyksen. Tämä on helpoin ja yksinkertaisin deinterlacing-skenaario, joka on yleinen sekä reaaliaikaisissa että jälkituotannon tehtävissä.

Jos alkuperäisen videokuvan päivitysnopeus on alle puolet lomitetun videon päivitysnopeudesta ja lomitetun videon päivitysnopeus on jaollinen alkuperäisen videon päivitysnopeudella, kaikki alkuperäiset kehykset ovat palautettavissa, mutta videossa on tietyin väliajoin ylimääräisiä kenttiä, jotka tulisi siistiä pois (kuva 7).



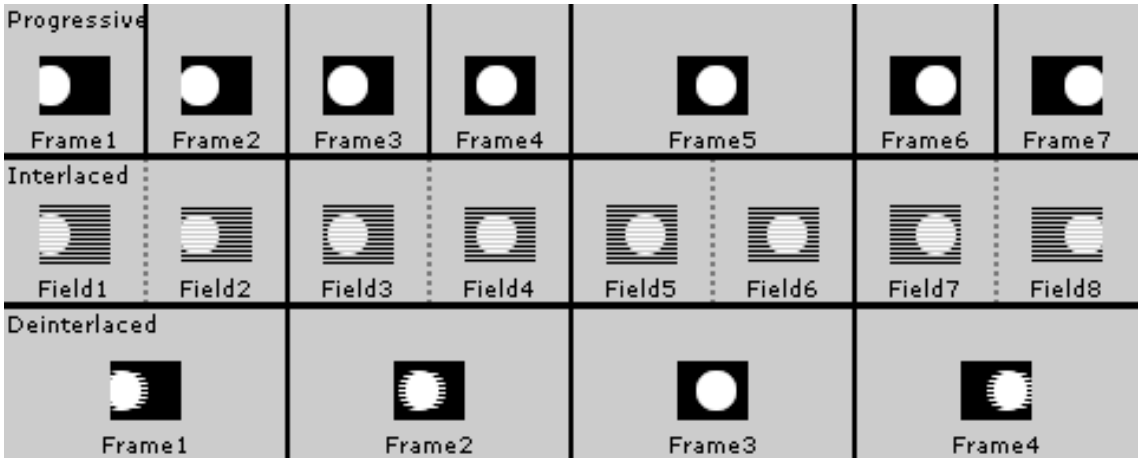
KUVA 7. Virheellinen lomituksen poisto, jossa alkuperäinen video 20 Hz.

Jos lomitetun videon päivitysnopeus ei ole jaollinen alkuperäisen videon päivitysnopeudella, alkuperäisen videon lomitusmuunnoksessa on yleensä käytetty erikoistahdistusta, jotta kuva ei repeytyisi (engl. screen tearing). Repeytymisen aiheuttaisi tasaisella tahdilla muuntaminen. Tällainen voi olla esimerkiksi 3:2 pulldown -muunnettu 24 Hz:n elokuva, jossa kehykset on tahdistettu siten, että kehyksistä luodaan vuorotellen kaksi tai kolme kenttää, jotta elokuvaa voitaisiin toistaa oikealla nopeudella 60 Hz:n televisiolla (kuva 8). Jotta lomituksen poisto tehtäisiin oikein, tulee samoihin kehykseen kuuluvat kentät yhdistää toisiinsa.



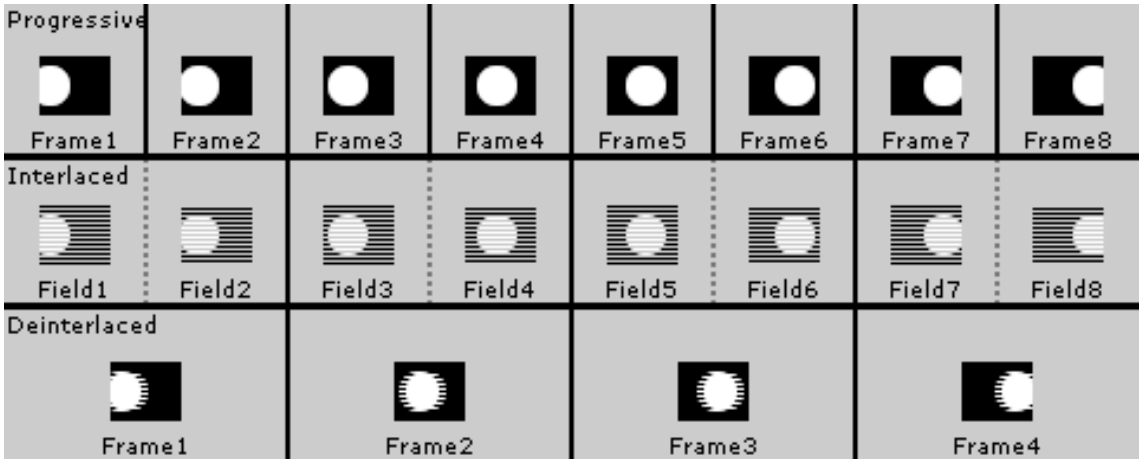
KUVA 8. 3:2 pulldown ja tyypillinen virheellinen lomituksen poisto.

Jos alkuperäisen videon päivitysnopeus on yli puolet lomitetun videon päivitysnopeudesta, kaikkia alkuperäisiä kehyksiä ei voida palauttaa ja suuresta osasta kehyksiä tulee virheellisiä. Lomitusmuunnos on yleensä tahdistettu siten, että joistakin kehyksistä on tehty useampia kenttiä, jotta kuva ei repeytyisi (kuva 9).



KUVA 9. Virheellinen lomituksen poisto, jossa alkuperäinen video 50 Hz. Joka 5:s kehys näytetään kahdesti, jotta video toistettaisiin oikein 60 Hz:n televisiolla.

Jos alkuperäinen progressiivinen video päivittyy samaan tahtiin kuin lomitettu video tai jos video on tallennettu alun perin lomitettuna, vierekkäisten kenttien yhdistäminen johtaa siihen, että kaikista kehyksistä tulee virheellisiä (kuva 10).



KUVA 10. Virheellinen lomituksen poisto, jossa sekä alkuperäinen että lomitettu video 60 Hz.

Jos taas alkuperäisen progressiivisen videon päivitysnopeus on nopeampi kuin lomitetun videon, joitakin kehyksiä on jouduttu heittämään kokonaan pois, jotta lomitusmuunnos on voitu edes toteuttaa.

Kahdesta erinäköisestä aikahetkestä otettujen kenttien yhdistäminen johtaa hiuskamman näköisiin combing-artefakteihin. Jotta tältä vältyttäisiin, on olemassa useita erilaisia deinterlacing-algoritmeja, joiden tavoitteena on yrittää päätellä puuttuvien kenttien vaakajuovien pikselien arvot. Deinterlacing-algoritmeja on reaaliaikaisiin ja jälkituotannon käyttötarkoituksiin.

Reaaliaikaisten deinterlacing-algoritmien tavoitteena on mahdollisimman nopea signaalin muunnos, jotta lomitettua signaalia voitaisiin käyttää heti. Reaaliaikaisia deinterlacing-algoritmeja on esim. line doubling, joka piirtää tyhjille vaakariveille kopion edellisestä vaakajuovasta eli jokainen vaakajuova piirretään kahdesti.

Jälkituotantoa varten on olemassa monimutkaisempia algoritmeja, jotka mahdollistavat laadukkaamman lomituksen poiston. Nämä algoritmit hyödyntävät kaiken saatavissa olevan laskentatehon päättelemään puuttuvien vaakajuovien pikselit muihin juoviin perustuen.

2.7 Värisignaalien perusteet

Väritelevisio piirtää kuvan kolmella erivärisellä valolla: punainen, vihreä ja sininen. Nämä ovat väritelevision käyttämät päävärit, joiden kirkkautta säätämällä ja toisiinsa sekoittamalla voidaan luoda kaikki näkyvät värit. Värillisen videosignaalin toteuttamiseksi on kehitetty useita erilaisia menetelmiä, joiden tarkoituksena on toteuttaa näiden kolmen värin informaatio sähköisesti.

2.7.1 RGB-video

RGB-video (Red Green Blue) käyttää kolmea eri videosignaalia kuvan luomiseen. Jokainen videosignaali vastaa omasta väristään, jotka ovat punainen, vihreä ja sininen. Televisio piirtää nämä kolme värisignaalia samaan kohtaan, jolloin nämä signaalit muodostavat yhdessä halutun värin. Sync-signaalit voivat olla upotettu johonkin kolmesta värisignaalista (yleensä vihreä) tai sync-signaaleille on omat johtimet.

RGB-video on yksinkertaisin ja virheettömin tapa toteuttaa värillinen videosignaali. RGB-signaali vaatii paljon kaistaa kolmelle erilliselle signaalille, joten se ei sovellu kaistarajoitteisiin sovelluksiin kuten televisiolähetyksiin.

2.7.2 Komponenttivideo

Komponenttivideo (engl. component video) rakentuu kolmesta komponentista: kirkkaussignaali Y (luma) ja värierosignaalit Pb ja Pr. Värierosignaaleja kutsutaan yhdessä kromaksi (engl. chroma).

Luma-signaali (Y) luodaan summaamalla RGB-värisignaalit toisiinsa omilla kertoimillaan: Y = (Kr * R) + (Kg * G) + (Kb * B) (8).

Värierosignaali Pb luodaan kertomalla omalla kertoimella sinisen RGB-signaalin ja luma-signaalin ero: Pb = Kcb * (B − Y) (8).

Värierosignaali Pr luodaan kertomalla omalla kertoimella punaisen RGB-signaalin ja luma-signaalin ero: Pr = Kcr * (R − Y) (8).

Käytetty televisiojärjestelmä määrittää kertoimien arvot (8).

Komponenttivideon värierosignaaleista voidaan suodattaa ylimääräiset taajuudet pois kaistankäytön pienentämiseksi ilman silmin havaittavaa kuvanlaadun heikkenemistä (8).

Komponenttivideota kutsutaan myös nimellä YUV ja YPbPr (9, hakusana YPbPr).

2.7.3 S-video

S-videosignaali rakentuu kahdesta komponentista: kirkkaussignaali Y (luma) ja värisignaali C (chroma).

Chroma-signaali luodaan vaihe- tai taajuusmoduloimalla komponenttivideon Pb- ja Pr-värierosignaalit, värille tarkoitetuilla apukantoaallosignaaleilla ja yhdistämällä nämä moduloidut signaalit toisiinsa yhdeksi värisignaaliksi. Tämä värisignaali myös kaistanpäästösuodatetaan kaistankäytön pienentämiseksi. Luma-signaali on sama kuin komponenttivideossa. (8.)

S-videon chroma-signaalin luontiprosessi laskee videon kuvanlaatua hieman komponenttivideota huonommaksi.

S-videota kutsutaan joskus nimellä Y/C (9, hakusana s-video).

2.7.4 Komposiittivideo

Värikomposiittivideo kehitettiin analogisia televisiolähetyksiä varten. Televisiolähetykset olivat hyvin kaistarajoitteisia, joten värillinen videosignaali oli pystyttävä lähettämään yhdellä kantataajuudella. Lisäksi väritelevisiosignaalin piti olla taaksepäin yhteensopiva mustavalkotelevisioiden kanssa, jotka käyttivät pelkästään luma-signaalia.

Tämän saavuttamiseksi komposiittivideo (engl. composite video) yhdistää chroma- ja luma-signaalit yhdeksi signaaliksi, jotta se voidaan lähettää samalla tavalla kuin mustavalkotelevisiosignaalikin. Mustavalkotelevisio vastaanottaa ja näyttää komposiittivideossa olevan luma-signaalin ja jättää chroma-signaalin huomioimatta. Väritelevisio tunnistaa chroma-signaalin komposiittivideosignaalista ja määrittää oikean värin sen avulla.

Komposiittivideon kuvanlaatu on huonompi kuin S-videon, sillä chroma- ja luma-signaalien yhdistäminen yhdeksi signaaliksi johtaa siihen, että niitä ei voida enää erottaa toisistaan puhtaasti.

Komposiittivideota kutsutaan myös nimellä CVBS (composite video baseband signal) (9, hakusana cvbs).

Yleisimmät analogiset videoliitännät:

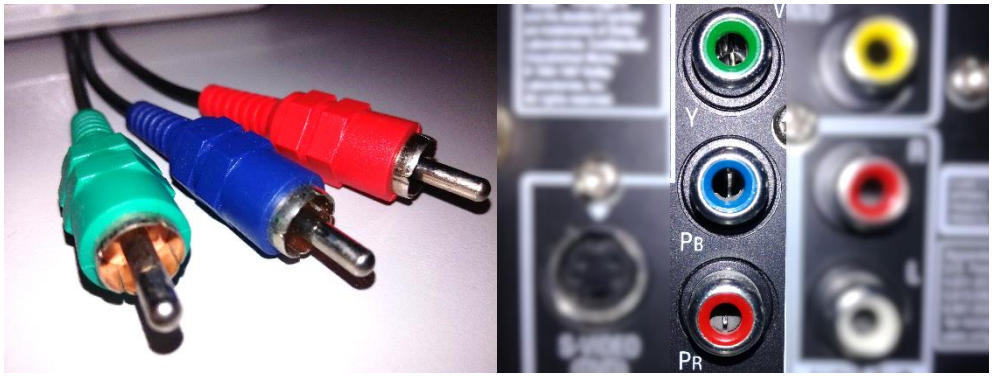
- Composite video (CVBS)
    - 1 RCA-liitäntä (kuva 11)
    - Kaapelit ja liitännät on yleensä värikoodattu keltaisella värillä

KUVA 11. Komposiittivideokaapeli ja liitäntä.

- S-video (Y/C)
    - 4-pinninen mini-DIN-liitäntä (kuva 12)
        1. Ground Y (Luma)
        2. Ground C (Chroma)
        3. Y (Luma)
        4. C (Chroma)

KUVA 12. S-videokaapeli ja liitäntä.

- Component video (YPbPr)
    - 3 RCA-liitäntää (kuva 13)
    - Kaapelit ja liitännät on yleensä värikoodattu
        - Y: vihreä
        - Pb: sininen
        - Pr: punainen (ei saa sekoittaa oikean stereokanavan punaisen RCA-liitännän kanssa)



KUVA 13. Komponenttivideokaapelit ja liitännät.

- SCART
    - 21-pinninen liitäntä (kuva 14)
    - Tukee useita analogisia videosignaaleja
        - RGB
        - Komposiittivideo
        - S-video
        - Komponenttivideo (signaalia voi kuljettaa liitännän kautta, mutta suurin osa SCART-liitännällä varustetuista televisioista ei tue komponenttivideosignaalia SCART-liitännän kautta)
    - Tukee myös stereoääntä videon kanssa samassa kaapelissa

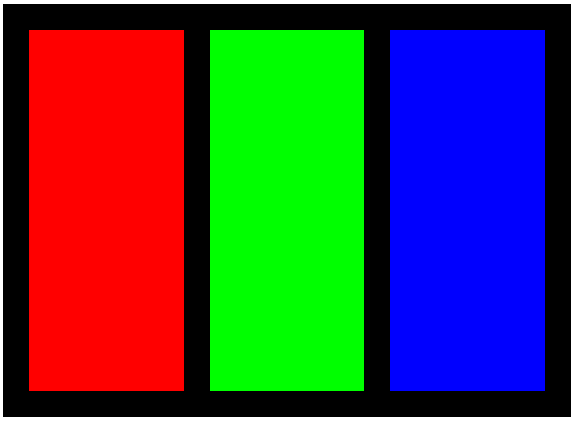
KUVA 14. SCART-videokaapeli ja liitäntä.

- VGA
    - 15-pinninen DE-15-liitäntä (kuva 15)
    - Toimii RGB-videosignaaleilla
    - Käytetään enimmäkseen tietokonemonitoreissa

KUVA 15. VGA-videokaapeli ja liitäntä.

2.8 Pikseli

Nykyään näytöissä ei käytetä liikkuvaa elektronisädettä, vaan näytöt rakentuvat tietystä määrästä pikseleitä, joita ohjataan erikseen. Pikselit rakentuvat pienistä värielementeistä ja jokaiselle pikselille on yleensä oma punainen (R), vihreä (G) ja sininen (B) värielementti (kuva 16). Kun näiden kolmen lähekkäin sijoitetun värielementin kirkkauksia säädetään, ne luovat yhdessä illuusion tasaisesta väristä.



KUVA 16. Tyypillisen pikselin RGB-elementit.

Koska analogiset videosignaalit oli alun perin suunniteltu kuvaputkitelevisioille, joissa ei ole tiettyä määrää valmiiksi asetettuja pikseleitä, analogista videosignaalia ei voida käyttää suoraan kuvan muodostamiseen, vaan videosignaali on muutettava ensin digitaaliseen muotoon pikseleiden RGB-elementtien arvoiksi. Tämä tapahtuu siten, että videosignaalia näytteistetään korkealla tahdilla ja kerätyistä näytteistä lasketaan pikselien väriarvot digitaalisesti.

3 TIETOKONEEN ÄÄNILAITTEISTO

Tietokoneesta löytyy monenlaisia liitäntöjä eri käyttötarkoituksiin. Suurin osa näistä on kuitenkin digitaalisia liitäntöjä, jotka eivät sovellu analogisten signaalien kaappaamiseen ilman erillistä D/A-muunninta. Tietokoneen analogisista liitännöistä yleensä löytyvät ääni- ja VGA-liitännät. VGA-liitäntä on tarkoitettu analogisen videosignaalin generointiin, mutta tämän liitännän ulostulon tarkka ohjaaminen vaatii matalan tason ohjelmointia (low-level programming). Ääniliitäntöjä voidaan käyttää analogisen signaalin kaappaukseen ja generointiin.

Videosignaalia olisi myös mahdollista generoida tietokoneen digitaalisten ulostuloliitäntöjen kautta, mutta näille liitännöille ei ole saatavilla RCA-videoliitäntään sopivaa adapteria, joten niiden käyttäminen olisi hieman työläämpää kuin ääniliitäntöjen käyttö.

3.1 Ääniliitännän ominaisuudet

Tietokoneen ääniliitännän käyttö ei vaadi matalan tason ohjelmointia ja sitä voidaan hallita vaikkapa verkkosivuohjelmointiin tarkoitetulla JavaScriptillä. Tästä syystä se on yksi helpoimmista ja yksinkertaisimmista tavoista kaapata ja generoida analogista signaalia. Tietokoneen ääniliitännät ovat yleensä 3,5 mm:n ääniliitäntöjä, ja tällainen liitäntä voidaan sovittaa helposti komposiittivideon käyttämäksi RCA-liitännäksi yksinkertaisen ja halvan adapterin avulla.

3.2 Äänilaitteiston signaalinkäsittelyominaisuudet

Nykyaikaisten tietokoneiden emolevyille integroidussa äänilaitteistossa on usein yllättävän joustava A/D- ja D/A-muunnin. Näiden muuntimien signaalin vahvistusta ja näytteistystaajuutta voidaan säätää ohjelmallisesti. Tietokoneen äänilaitteiston näytteistystaajuudet sijoittuvat yleensä alueelle 44,1–192 kHz riippuen käytetystä äänipiiristä. Nyquistin teoreema määrittää, että X Hz:n taajuisen signaalin kaappaamiseen tarvitaan vähintään 2 kertaa X:n kokoinen näytteistystaajuus. Esimerkiksi 192 kHz:n näytteistystaajuuden sisääntulon pitäisi Nyquistin teoreeman mukaan pystyä kaappaamaan maksimissaan 96 kHz:n taajuisia signaaleja.

4 VIDEON KAAPPAUS JA GENEROINTI

Videokuvan kaappaus ja generointi toteutettiin oman testikoneeni integroidulla äänipiirillä, joka kykeni 192 kHz:n näytteistystaajuuden tallennukseen ja toistoon.

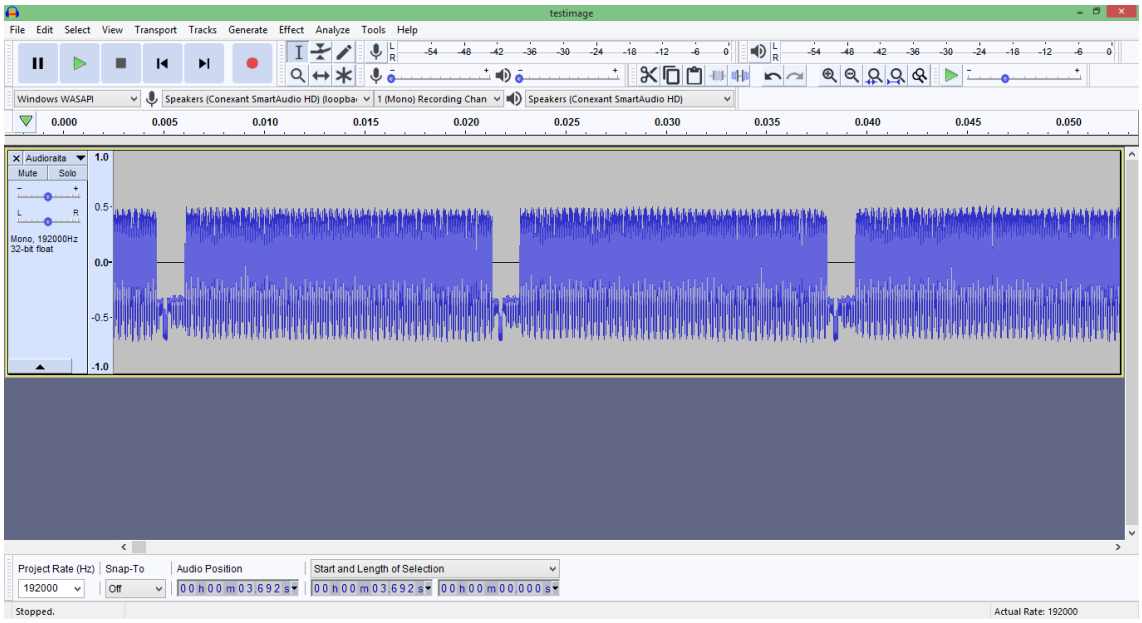
4.1 Videokuvan kaappaus

Tein JavaScript-ohjelman (liite 1), joka analysoi kaapatun videosignaalin näytteet ja piirtää niihin perustuen kuvan.

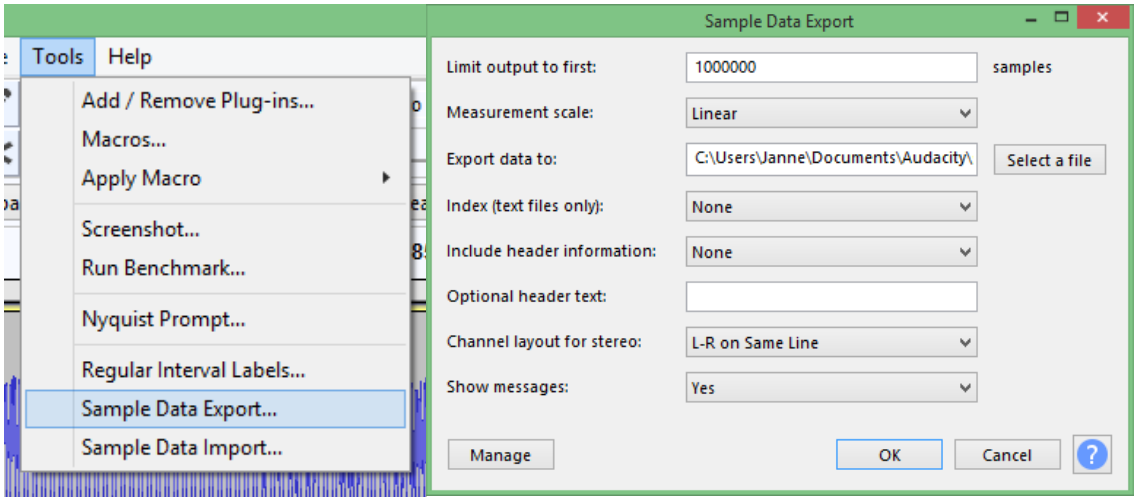
Lähetin testikuvan videosignaalia PSP-3000-pelikonsolin komponenttivideo-kaapelin kautta tietokoneen line in -äänituloon. Komponenttivideokaapelin vihreä johto kuljettaa videosignaalin kirkkausnäytteitä (luma), jotka vastaavat harmaasävyistä komposiittivideosignaalia. Kytkin tämän RCA-johdon adapterin avulla tietokoneen 3,5 mm:n line in -äänituloon. Laitteet näkyvät kuvassa 17.

KUVA 17. PSP-3000-pelikonsoli ja sen komponenttivideokaapelit (vasemmalla) ja 3,5 mm – RCA -adapteri (oikealla).

Tallensin testikuvan (kuva 20) videosignaalia Audacity-ohjelmalla (kuva 18) ja vein tallennetut näytteet tekstitiedostoon ”Sample Data Export”-työkalulla (kuva 19). Olisin voinut tallentaa videosignaalia suoraan JavaScript-ohjelmallakin, mutta en kokeillut tätä ajanpuutteen vuoksi.



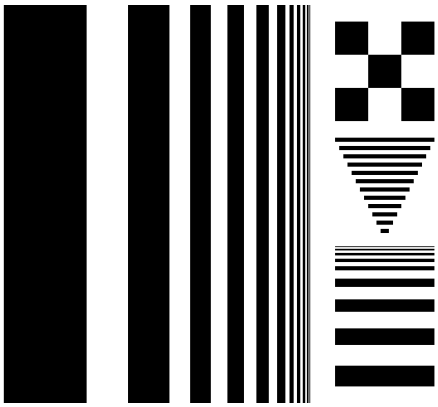
KUVA 18. Testikuvan videosignaalin tallenne avattuna Audacity-äänenkäsittely-ohjelmassa (https://www.audacityteam.org/).



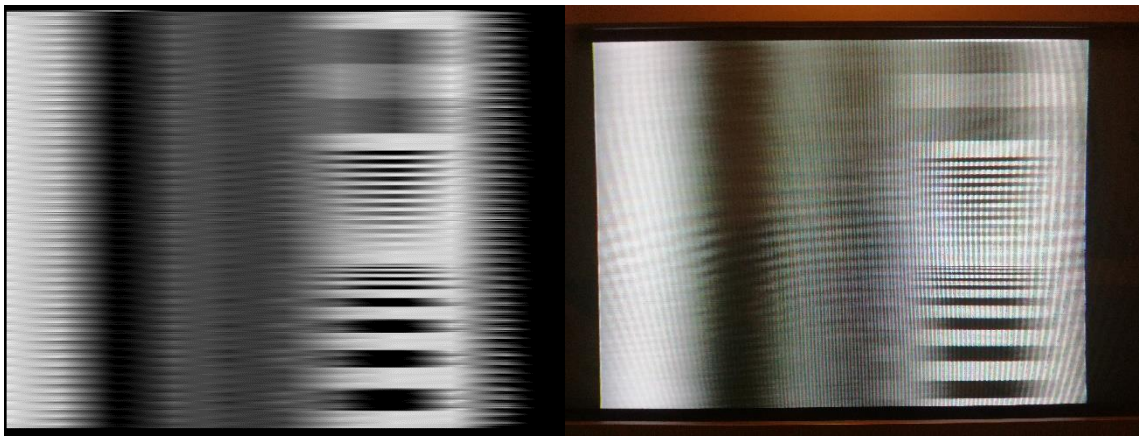
KUVA 19. Audacity-ohjelman Sample Data Export -työkalu. Signaalin näytteet vietiin tekstitiedostoon kuvassa näkyvin asetuksin.

Muunsin tekstitiedostossa olevat näytteet JavaScript-taulukoksi, jonka latasin videonkaappausohjelmaan. Tein ohjelman tunnistamaan näytteistä sync-pulssit ja pikselien kirkkausarvot. Ajanpuutteen vuoksi en ehtinyt ohjelmoimaan sync-pulssien jännitetason automaattista tunnistamista, joten jännitetaso pitää asettaa manuaalisesti koodiin.

Kun kaikki sync-pulssit ja kirkkausarvot on tunnistettu, ohjelma piirtää ne pikselimuodossa HTML5 canvas -elementtiin, joka vastaa sitä, miltä kuva näyttäisi television ruudulla (kuva 21).



KUVA 20. Alkuperäinen testikuva.



KUVA 21. Javascript-ohjelmalla tehty videosignaalin piirros (vasemmalla) ja sama kaapattu signaali toistettuna television ruudulta(oikealla).

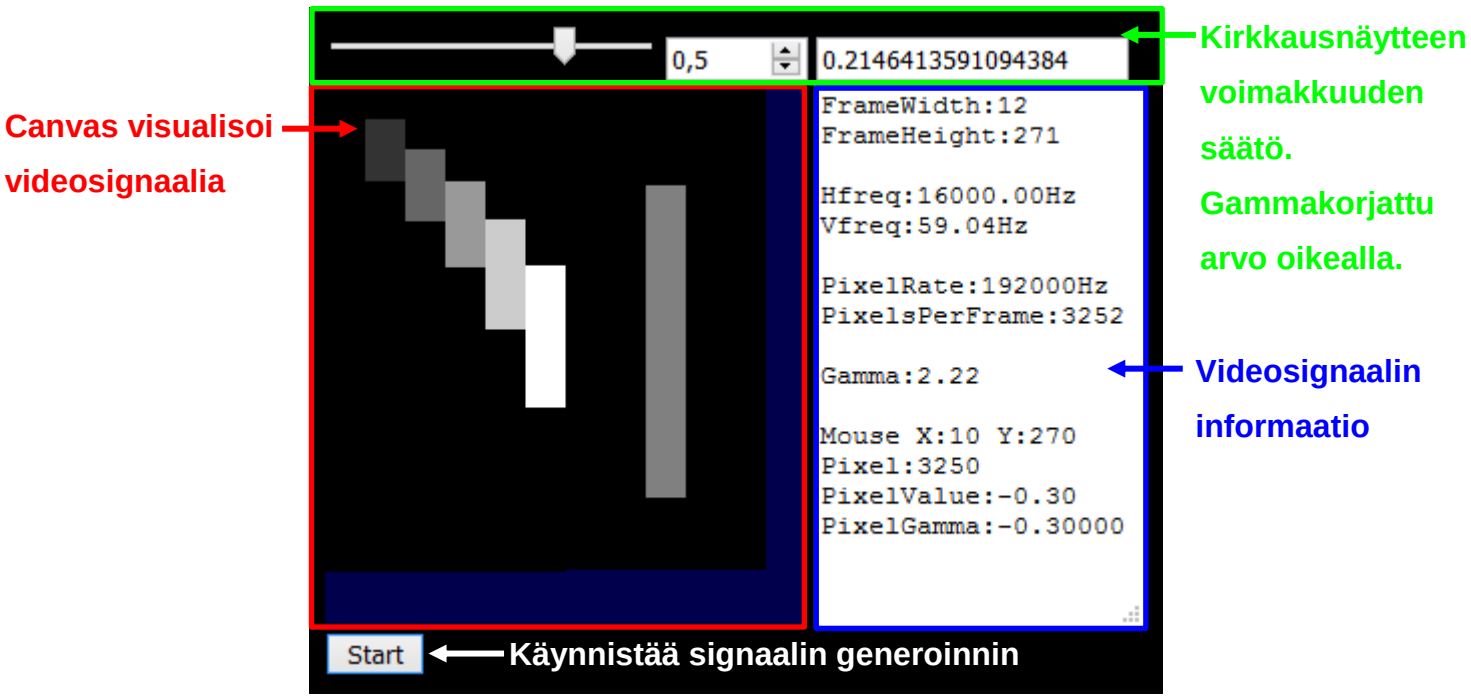
Pieni näytteistystaajuus johtaa siihen, että kaapattu kuva on hyvin sumea. Pitkät vaakasuuntaiset viivat on vielä mahdollista erottaa kuvasta, koska niistä on saatu useita näytteitä samalta juovalta. Kapeat pystysuuntaiset viivat sekoittuvat toisiinsa, sillä niistä ei ole mahdollista saada tarpeeksi näytteitä.

Komposiittivideosignaalin jännitetaso muuttuu miljoonia kertoja sekunnissa, joten signaalin kaappaaminen alkuperäisessä muodossaan on mahdotonta. 192 kHz:n näytteistämisellä on kuitenkin mahdollista kaapata kaikki 60 Hz:n videosignaalin Hsync- ja Vsync-pulssit sekä noin 10 pikselinäytettä jokaiselta juovalta. Värikuvan kaappaaminen vaatisi vielä nopeamman näytteistystaajuuden värisignaalin (chroma) korkeamman taajuuden vuoksi.

4.2 Videokuvan generointi

Tein JavaScript-ohjelman (liite 2), joka generoi Hsync- ja Vsync-pulsseja sisältävää videosignaalia tietokoneen line out -äänilähdöstä. Videongenerointiohjelma luo äänipuskurin, johon asetetaan negatiivisia näytteitä sync-pulssien mukaisiin aikahetkiin. Äänipuskurin loput näytteet alustetaan nollilla ja tuloksena on musta kuva. Tämän jälkeen äänipuskurissa olevaa videosignaalia voidaan toistaa tietokoneen äänilähdöstä, television komposiittivideotulon kautta ruudulle.

Käyttäjä voi lisätä videosignaaliin kirkkausnäytteitä piirtämällä pikseleitä generointiohjelman Canvas-elementtiin. Canvas on kuvanpiirtoalue-elementti, jota käytetään tässä ohjelmassa visualisoimaan generoitavaa videosignaalia. Sync-pulssit näkyvät kuvassa sinisellä värillä ja kirkkausnäytteet harmaasävyillä. Äänipuskuria päivitetään reaaliaikaisesti canvakseen piirretyillä uusilla näytteillä. Äänipuskuriin lisätyt kirkkausnäytteet on gammakorjattu NTSC-televisiojärjestelmän mukaisella 2,22 gammakertoimella, jotta television ruudulla näkyvä kuva täsmäisi generointiohjelmassa näkyvän kuvan kanssa. Käyttöliittymä näkyy kuvassa 22.



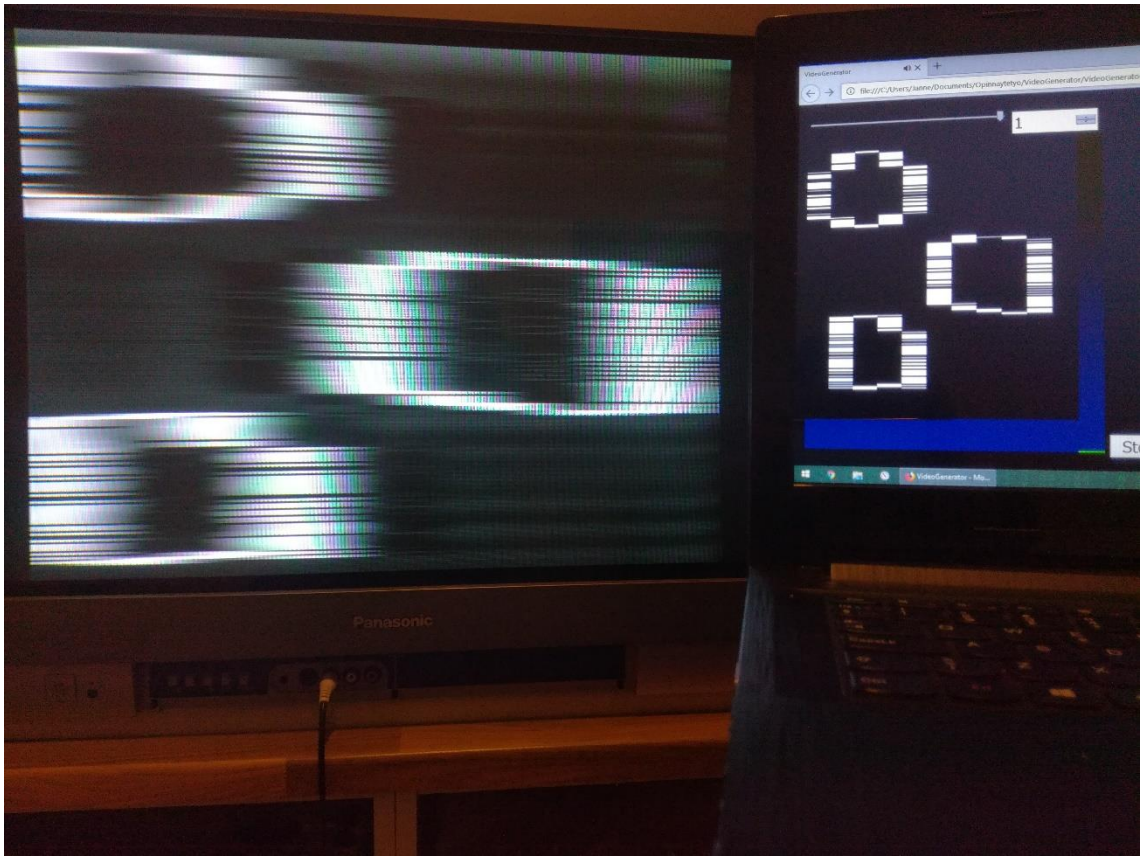
KUVA 22. Videokuvan generointiohjelma.

Koska äänilähdön näytteistystaajuus oli 192 kHz, valitsin juovanpiirtotaajuudeksi 16 kHz, jolloin Hsync-pulssit asetetaan tasan 12 näytteen välein (192000 / 16000

= 12). Asetin kuvan päivitystaajuudeksi 59,04 Hz määrittämällä näytteiden määräksi 3252 per kenttä (192000 / 3252 ≈ 59,04). Valitsin kyseisen taajuuden, koska käyttämäni televisio näytti vakaimman kuvan tällä päivitystaajuudella.

Tietokoneen äänilähtö kytkettiin television komposiittivideotuloon käyttäen samaa 3,5 mm – RCA -adapteria ja RCA-johtoa kuin videon kaappauksessakin. Tietokoneen äänenvoimakkuutta säätämällä voidaan hallita koko videosignaalin voimakkuutta. Äänenvoimakkuus määrittää äänilähdöstä tulevan signaalin jännitetason, jota muuttamalla voidaan parantaa signaalin tunnistamista televisiossa. Videongenerointiohjelma toimii Firefox-verkkoselaimella.

Vaikka kuvan vaakaresoluutio on hyvin pieni (11 näytettä + Hsync-pulssi), se riittää yksinkertaisten kuvioiden piirtämiseen. Osa kuvasta menee overscan-alueelle, mutta suurin osa kuvasta jää näkyviin. Kenttien 271 vaakajuovasta noin 200 juovaa on näkyvissä. Videongenerointiohjelman toiminta näkyy kuvassa 23.



KUVA 23. Videokuvan generointiohjelma (oikealla) lähettämässä videosignaalia televisiolle (vasemmalla).

5 POHDINTA

Työn tavoitteena oli oppia analogisten videosignaalien toiminta ja laatia siitä tämä raportti. Tavoitteena oli myös testata voiko tietokoneesta löytyvillä liitännöillä kaapata ja generoida analogisia videosignaaleja.

Aloitin tämän opinnäytetyön tekemisen kesällä 2018 selvittämällä analogisten videosignaalien perusteita ja kehittämällä prototyypin videonkaappausohjelmasta. Ohjelmaa kehittäessä opin paljon analogisten videosignaalien ja kuvaputkitelevisioiden toiminnasta.

Videosignaalin kaappausohjelma oli hyvin kokeellinen ja se muuttui monta kertaa kehityksen aikana. Kokeilin useita keinoja parantaa kuvan laatua, mutta äänilaitteiston liian pieni näytteistystaajuus koitui aina ongelmaksi.

Laadukkaan liikkuvan kuvan kaappaaminen pienellä näytteistystaajuudella on mahdotonta, sillä suurin osa videosignaalin informaatiosta häviää, mutta jos kuva pysyy staattisena, laadukkaampi keskiarvoistettu kaappaaminen saattaa olla mahdollista. Itse en saanut tätä toimimaan (liite 3).

Videosignaalin generointiohjelma onnistui mielestäni hyvin. Tietokoneen äänilähdön kautta oli mahdollista tuottaa tunnistettavissa oleva, harmaasävyinen komposiittivideosignaali. Alun perin videon generointiin oli tarkoitus kokeilla myös muitakin tietokoneen ulostuloja, kuten sarjaportti tai rinnakkaisportti. Näiden käyttö olisi tosin ollut monimutkaisempaa, joten hylkäsin idean ajanpuutteen vuoksi.

Raporttia kirjoittaessa ja kuvaajia piirtäessä tuli ilmi uusia yksityiskohtia, joiden avulla pystyin parantamaan videon kaappauksen ja generoinnin ohjelmia.

Opinnäytetyön päätavoitteet onnistuivat mielestäni hyvin. Opin paljon työtä tehdessä ja tästä uudesta tiedosta saattaa olla hyötyä tulevaisuudessa.

LÄHTEET

1. Virtual Console, mikä se on? 2018. Oy Bergsala AB/Nintendo Suomen virallinen verkkosivusto. Saatavissa: https://www.nintendo.fi/tuki/248-virtual-console. Hakupäivä 3.6.2019.

2. HD Retrovision Bringing retro gaming into the high definition era. HD Retrovision LLC. Saatavissa: https://www.hdretrovision.com/. Hakupäivä 3.6.2019.

3. marqs85/ossc. Open Source Scan Converter. GitHub. Saatavissa: https://github.com/marqs85/ossc. Hakupäivä 3.6.2019.

4. Open Source Scan Converter (OSSC). RetroRGB. Saatavissa: https://www.retrorgb.com/ossc.html. Hakupäivä 3.6.2019.

5. Analogue - Mega Sg. Analogue. Saatavissa: https://www.analogue.co/mega-sg/. Hakupäivä 3.6.2019.

6. Hi-Def NES. RetroRGB. Saatavissa: https://www.retrorgb.com/hidefnes.html. Hakupäivä 3.6.2019.

7. The story of BBC Television - How it all began. BBC. Saatavissa: https://www.bbc.co.uk/historyofthebbc/research/general/tvstory1. Hakupäivä 25.04.2019.

8. Understanding Analog Video Signals. 2002. Maxim Integrated. Saatavissa: https://www.maximintegrated.com/en/app-notes/index.mvp/id/1184. Hakupäivä 30.11.2018.

9. TEPA-termipankki. Sanastokeskus TSK. Saatavissa: http://www.tsk.fi/tepa/fi/. Hakupäivä 29.5.2019.

VideoCapture.html

```
<!doctype html>
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
<html>
<head>
<meta charset="utf-8">
<title>VideoCapture</title>
<script type="text/javascript" src="data4.js"></script>
<script type="text/javascript" src="VideoCapture.js"></script>
<!-- <script type="text/javascript" src="VideoCaptureTest.js"></script> -->
</head>
<body bgcolor="#000">
<canvas id="videoCanvas"></canvas>
</body>
</html>
```

VideoCapture.js

```
document.addEventListener("DOMContentLoaded", function(event) {
 document.addEventListener('keydown', function(event) {
  if(event.keyCode === 107) {
   imageWidth+=10;
   event.preventDefault();
   RenderVideo();
  } else if(event.keyCode === 109) {
   imageWidth-=10;
   event.preventDefault();
   RenderVideo();
  } else if(event.keyCode === 36) {
   ChangeBrightness(0.1);
   event.preventDefault();
  } else if(event.keyCode === 35) {
   ChangeBrightness(-0.1);
   event.preventDefault();
  }
 });
 var imageWidth = 640,imageHeight = 525;
 var imageLine = 0,imageFieldBufferSize = 10,imageFieldCount = -1;
 var imageBuffer, imageBufferData;
 var kerroin = 2;
 var interlaced = true; //false == 240p
 if(!interlaced){
  imageWidth = imageWidth/2;
  //imageHeight = imageHeight/2;
 }

 var sampleIndex = 0;
 var sampleLimit = samples.length;
 var sampleTimer = 0;

 var videoCanvas = document.getElementById('videoCanvas');
 videoCanvas.width = imageWidth;
 videoCanvas.height = imageHeight;
 var videoRenderArea = videoCanvas.getContext('2d');
 var scanlineBuffer = [];
 var emptyBuffer = false;
 var scanlineLength = 11, newScanlineLength = 0;
 var odd=false;

 var syncThreshold = -0.37;
```

```
 function RenderVideo(){
  imageBuffer = new Array(imageHeight);
  for(i=0;i<imageBuffer.length;i++){
   imageBuffer[i] = new Array(imageWidth);
   for(b=0;b<imageBuffer[i].length;b++){
    imageBuffer[i][b] = new Array(imageFieldBufferSize);
    imageBuffer[i][b].fill(0);
   }
  }
  videoCanvas.width = imageWidth;
  videoCanvas.height = imageHeight;
  videoRenderArea = videoCanvas.getContext('2d');

  sampleIndex = 0;
  var waitForNextScanline = 0;
  while(sampleIndex < sampleLimit){
   if(samples[sampleIndex] >= syncThreshold){ //sync-signalin taso (datasta voisi etsiä
synctason?)
    if(scanlineLength === 0)
     scanlineLength = newScanlineLength;
    if((waitForNextScanline > 0 && newScanlineLength/scanlineLength >= 0.9) ||
newScanlineLength/scanlineLength >= 1){
     scanlineLength += 0.01*(newScanlineLength-scanlineLength); //Lerp 1%
     newScanlineLength = 0;
     emptyBuffer = true;
    }
    waitForNextScanline = 0;
   } else { //samples[sampleIndex] < 0
    waitForNextScanline++;
   }
   if(!emptyBuffer){
    if(waitForNextScanline <= 5){
     scanlineBuffer.push(samples[sampleIndex]);
     newScanlineLength++;
    } else {
     if(odd || !interlaced){
      imageLine = 0; //vsync
      odd = false;
     } else{
      imageLine = 1;
      odd = true;
     }
     imageFieldCount++;
     if(imageFieldCount >= imageFieldBufferSize)
      imageFieldCount = 0;
    }
   } else {
    var distance = imageWidth/(scanlineBuffer.length-1);
    if(imageFieldCount > -1) //ignore first field
    for(i=0;i<imageWidth;i++)
    {
     var index = Math.floor(i/imageWidth*(scanlineBuffer.length-1)); //scanlineLength?
     var interpolant = (i-index*distance)/distance;
     if(index < scanlineBuffer.length-1)
      imageBuffer[imageLine][i][imageFieldCount] =
scanlineBuffer[index]+interpolant*(scanlineBuffer[index+1]-scanlineBuffer[index]); //Linear
interpolation
     else
      imageBuffer[imageLine][i][imageFieldCount] = scanlineBuffer[index];
    }
    if(interlaced)
     imageLine+=2;
    else
     imageLine+=1;
```

```
    emptyBuffer = false;
    scanlineBuffer = [];
   }
   sampleIndex++;
  }
  //DrawImageBuffer();
  DrawAvgImageBuffer();
  //videoRenderArea.fillStyle="#000000";
  //videoRenderArea.font="20px Arial";
  //videoRenderArea.fillText("Sample:"+sampleIndex,10,20);
  //videoRenderArea.fillText("SampleDivider:"+sampleDivider,10,40);
 }
 function DrawImageBuffer(){
  //videoRenderArea.clearRect(0, 0, videoCanvas.width, videoCanvas.height);
  for(y=0;y<imageBuffer.length;y++){
   for(x=0;x<imageBuffer[y].length;x++){
    videoRenderArea.fillStyle="rgba(255,255,255,"+(imageBuffer[y][x][0])+")";
    videoRenderArea.fillRect( x, y, 1, 1 );
   }
  }
  imageBufferData = videoRenderArea.getImageData(0,0,videoCanvas.width,videoCanvas.height);
  ChangeBrightness(0);
 }
 function DrawAvgImageBuffer(){
  //videoRenderArea.clearRect(0, 0, videoCanvas.width, videoCanvas.height);
  for(y=0;y<imageBuffer.length;y++){
   for(x=0;x<imageBuffer[y].length;x++){
    var avgLuma = 0;
    for(z=0;z<imageBuffer[y][x].length;z++){
     avgLuma += imageBuffer[y][x][z];
    }
    if(interlaced)
     avgLuma /= imageBuffer[y][x].length/2;
    else
     avgLuma /= imageBuffer[y][x].length;
    videoRenderArea.fillStyle="rgba(255,255,255,"+(avgLuma+Math.abs(syncThreshold)-0.17)+")";
    videoRenderArea.fillRect( x, y, 1, 1 );
   }
  }
  imageBufferData = videoRenderArea.getImageData(0,0,videoCanvas.width,videoCanvas.height);
  ChangeBrightness(0);
 }
 function ChangeBrightness(amount){
  kerroin+=amount;

  var imageBufferCopy = new ImageData(new
Uint8ClampedArray(imageBufferData.data),imageBufferData.width,imageBufferData.height);

  for(i=3; i<imageBufferCopy.data.length; i+=4){
   imageBufferCopy.data[i] = imageBufferCopy.data[i]*kerroin; //imageBufferCopy.data[i] =
Math.pow(imageBufferCopy.data[i],1/2.22)*kerroin;
  }
  videoRenderArea.putImageData(imageBufferCopy,0,0);
 }

 RenderVideo();
});
```

VideoGenerator.html

```
<!doctype html>
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
<html>
<head>
<meta charset="utf-8">
<title>VideoGenerator</title>
<script type="text/javascript" src="VideoGenerator.js"></script>
</head>
<body bgcolor="#000">
<div style="width:460px;">
<input type="range" min="-100" max="100" id="videoSlider">
<input type="number" step="0.01" min="-1" max="1" style="width:65px;" id="videoNumber">
<input type="text" style="width:150px;" id="outputNumber">
<canvas style="cursor: none;" id="videoCanvas" oncontextmenu="return false"></canvas>
<textarea style="width:160px; height:265px;" id="videoInfo"></textarea>
<button id="videoButton">Start</button>
</div>
</body>
</html>
```

VideoGenerator.js

```
document.addEventListener("DOMContentLoaded", function(event) {
 var samplesPerFrame = 3252, sampleRate = 192000;
 var videoWidth = 12, videoHeight = 271;

 widthMultiplier = 20;
 var videoCanvas = document.getElementById('videoCanvas');
 videoCanvas.width = videoWidth*widthMultiplier;
 videoCanvas.height = videoHeight;
 var videoRenderArea = videoCanvas.getContext('2d');
 videoRenderArea.scale(widthMultiplier,1);

 var videoButton = document.getElementById("videoButton");
 var videoOutputIsOn = false;

 var mouseX = 0, mouseY = 0, mouseDraw = false, mouseErase = false;
 var prevCursorX = 0, prevCursorY = 0;
 var videoPixelValue = 0.5;
 var gamma = 2.22;

 var videoNumber = document.getElementById("videoNumber");
 var outputNumber = document.getElementById("outputNumber");
 var videoSlider = document.getElementById("videoSlider");

 var videoInfo = document.getElementById("videoInfo");
 var mousePixel = 0;

 function ClampValue(value,min,max)
 {
  return Math.min(Math.max(value, min), max);
 }

 function UpdateVideoInfo()
 {
  videoInfo.value =
  "FrameWidth:"+videoWidth+"\n"+
  "FrameHeight:"+videoHeight+"\n\n"+
  "Hfreq:"+(sampleRate/videoWidth).toFixed(2)+"Hz\n"+
  "Vfreq:"+(sampleRate/samplesPerFrame).toFixed(2)+"Hz\n\n"+
```

```
 "PixelRate:"+sampleRate+"Hz\n"+
 "PixelsPerFrame:"+samplesPerFrame+"\n\n"+
 "Gamma:"+gamma+"\n\n"+
 "Mouse X:"+mouseX+" Y:"+mouseY+"\n"+
 "Pixel:"+(mouseX+mouseY*videoWidth)+"\n"+
 "PixelValue:"+(mousePixel > 0 ? Math.pow(mousePixel,1/gamma) : mousePixel).toFixed(2)+"\n"+
 "PixelGamma:"+mousePixel.toFixed(5);
}

function UpdateVideoNumbers()
{
 videoNumber.value = videoPixelValue;
 videoSlider.value = videoPixelValue*100;

 if(videoPixelValue > 0)
  outputNumber.value = Math.pow(videoPixelValue,gamma);
 else
  outputNumber.value = videoPixelValue;
}

videoNumber.oninput = function() {
 videoPixelValue = ClampValue(videoNumber.value, -1, 1);
 UpdateVideoNumbers();
}
videoSlider.oninput = function() {
 videoPixelValue = ClampValue(videoSlider.value/100, -1, 1);
 UpdateVideoNumbers();
}

var vsyncSampleCounter = 0;
var hsyncSampleCounter = 0;

var videoCtx = new AudioContext();
var videoBuffer = videoCtx.createBuffer(1, samplesPerFrame, sampleRate);
var videoSource;
var videoData = new Float32Array(samplesPerFrame);

function UpdateVideoData() {
 videoSource.buffer = videoBuffer; //You can change the buffer while it's being played?!
 window.requestAnimationFrame(function(){if(videoOutputIsOn)UpdateVideoData();});
 videoBuffer.copyToChannel(videoData, 0);
}

function StartVideoOutput() {
 videoSource = videoCtx.createBufferSource();
 videoSource.buffer = videoBuffer;
 videoSource.loop = true;
 videoSource.connect(videoCtx.destination);
 videoSource.start();

 UpdateVideoData();
}

videoButton.onclick = function() {
 if(!videoOutputIsOn) {
  videoOutputIsOn = true;
  videoButton.innerHTML = "Stop";

  StartVideoOutput();
 } else {
  videoOutputIsOn = false;
  videoButton.innerHTML = "Start";

  videoSource.stop();
```

```
  }
 }
 videoCanvas.addEventListener('mousemove', function(event) {
  mouseX = ClampValue(Math.floor((event.pageX-this.offsetLeft)/widthMultiplier), 0, videoWidth-
1);
  mouseY = ClampValue(Math.floor(event.pageY-this.offsetTop), 0, videoHeight-1);

  MoveCursorPixel(mouseX,mouseY);
  if(mouseDraw || mouseErase)
   ChangeVideoPixel(mouseX,mouseY,!mouseErase);
 });

 videoCanvas.addEventListener('mousedown', function(event) {
  if(event.button == 0) {
   mouseDraw = true;
  } else {
   mouseErase = true;
  }
  ChangeVideoPixel(mouseX,mouseY,!mouseErase);
 });
 document.addEventListener('mouseup', function(event) {
  if(event.button == 0) {
   mouseDraw = false;
  } else {
   mouseErase = false;
  }
 });
 document.addEventListener('wheel', function(event) {
  videoPixelValue = ClampValue(videoPixelValue-Math.sign(event.deltaY)/100, -1, 1);
  UpdateVideoNumbers();
  MoveCursorPixel(mouseX,mouseY);
 });

 function SetVideoRenderFillStyle(newVideoPixelValue)
 {
  if(newVideoPixelValue > 0)
   videoRenderArea.fillStyle =
"rgba("+newVideoPixelValue*255+","+newVideoPixelValue*255+","+newVideoPixelValue*255+",1)";
  else
   videoRenderArea.fillStyle = "rgba(0,0,"+(-newVideoPixelValue*255)+",1)";
 }

 function MoveCursorPixel(x,y)
 {
  var prevVideoPixelValue = videoData[prevCursorX+prevCursorY*videoWidth];

  if(prevVideoPixelValue > 0)
   SetVideoRenderFillStyle(Math.pow(prevVideoPixelValue,1/gamma));
  else
   SetVideoRenderFillStyle(prevVideoPixelValue);

  videoRenderArea.fillRect(prevCursorX, prevCursorY, 1, 1);

  SetVideoRenderFillStyle(videoPixelValue);
  videoRenderArea.fillRect(x, y, 1, 1);

  prevCursorX = x;
  prevCursorY = y;

  mousePixel = videoData[x+y*videoWidth];
  UpdateVideoInfo();
 }

 function ChangeVideoPixel(x,y,pixelOn)
```

```
{
 if(pixelOn) {
  SetVideoRenderFillStyle(videoPixelValue);
  videoRenderArea.fillRect(x, y, 1, 1);

  if(videoPixelValue > 0)
   videoData[x+y*videoWidth] = Math.pow(videoPixelValue,gamma);
  else
   videoData[x+y*videoWidth] = videoPixelValue;
 } else {
  videoRenderArea.fillStyle = "#000000";
  videoRenderArea.fillRect(x, y, 1, 1);

  videoData[x+y*videoWidth] = 0;
 }

 mousePixel = videoData[x+y*videoWidth];
 UpdateVideoInfo();
}
function InitializeVideoPixels()
{
 for(var x = 0; x < videoWidth; x++) {
  for(var y = 0; y < videoHeight; y++) {
   var newVideoPixelValue = videoData[x+y*videoWidth];

   if(newVideoPixelValue > 0)
    SetVideoRenderFillStyle(Math.pow(newVideoPixelValue,1/gamma));
   else
    SetVideoRenderFillStyle(newVideoPixelValue);

   videoRenderArea.fillRect(x, y, 1, 1);
  }
 }
}

function InitializeSyncData() {
 var syncData = videoBuffer.getChannelData(0);
 for(var i = 0; i < videoBuffer.length; i++) {
  vsyncSampleCounter++;
  if(vsyncSampleCounter > samplesPerFrame-318) { //very simple vsync
   syncData[i] = -0.3;
   if(vsyncSampleCounter === samplesPerFrame) {
    vsyncSampleCounter = 0;
   }
  } else {
   hsyncSampleCounter++;
   if(hsyncSampleCounter === videoWidth) { //very simple hsync
    syncData[i] = -0.3;
    hsyncSampleCounter = 0;
   } else {
    syncData[i] = 0;
   }
  }
  videoData[i] = syncData[i];
 }
}
InitializeSyncData();
InitializeVideoPixels();
UpdateVideoNumbers();
UpdateVideoInfo();
});
```

VideoCaptureTest.js (toimii VideoCapture.html tiedoston kautta)

(Huom. keskeneräistä koodia, joka ei toimi oikein)

```
document.addEventListener("DOMContentLoaded", function(event) {
 document.addEventListener('keydown', function(event) {
  if(event.keyCode === 187) {
   imageWidth+=1;
   Reset();
   event.preventDefault();
  } else if(event.keyCode === 189) {
   imageWidth-=1;
   Reset();
   event.preventDefault();
  } else if(event.keyCode === 107) {
   imageHeight+=1;
   Reset();
   event.preventDefault();
  } else if(event.keyCode === 111) {
   imageHeight-=1;
   Reset();
   event.preventDefault();
  } else if(event.keyCode === 39) {
   pixelClock+=1;
   Reset();
   event.preventDefault();
  } else if(event.keyCode === 37) {
   pixelClock-=1;
   Reset();
   event.preventDefault();
  }else if(event.keyCode === 33) {
   pixelClock+=1000;
   Reset();
   event.preventDefault();
  } else if(event.keyCode === 34) {
   pixelClock-=1000;
   Reset();
   event.preventDefault();
  } else if(event.keyCode === 38) {
   speed+=1;
   event.preventDefault();
  } else if(event.keyCode === 40) {
   speed-=1;
   event.preventDefault();
  }
 });
 var imageWidth = 800,imageHeight = 525/2, speed = 100000;

 var videoCanvas = document.getElementById('videoCanvas');
 videoCanvas.width = imageWidth;
 videoCanvas.height = imageHeight;
 var videoRenderArea = videoCanvas.getContext('2d');
 videoRenderArea.fillStyle="rgba(255,255,255,1)";

 var sampleIndex=0;
 var sampleTimer=0;
 var sampleCounter=0;
 var sampleCounterLine=0;
 var hsync=1/15750;
 var vsync=1/59.94; //tämän pitää olla oikein, jotta ei olisi driftiä!
 var pixelClock=9015800; //pixelClockia ei voi laskea?
 var timeStep=1/pixelClock;
```

```
 var sampleStep=1/192000;
 //var pixelStep=sampleStep/timeStep; //montako pikseliä on ehtinyt siirtyä samplesta toiseen??? (tämä
lisätään x-muuttujaan)
 var index=0;
 //var odd = false; ei vielä, mutta kohta
 var x=0; //lisätään 1 kunnes vsync
 var y=0; //nollaan kun hsync (tai odd/even)
 function RenderVideo(){
  //videoRenderArea.clearRect(0, 0, videoCanvas.width, videoCanvas.height);
  for(i=0;i<speed;i++){

   sampleTimer+=timeStep;
   sampleCounter+=timeStep;
   sampleCounterLine+=timeStep;
   if(sampleTimer >= sampleStep){
    videoRenderArea.fillStyle="rgba(255,255,255,"+samples[sampleIndex]*2+")";
    videoRenderArea.fillRect( x, y, 1, 1 );
    sampleTimer=0;
    sampleIndex++;
   } else if(sampleCounter >= vsync){
    x=0;
    //if(odd)
     y=0;
    //else
     //y=1;
    sampleCounter=0;
    sampleCounterLine=0;
   } else if(sampleCounterLine >= hsync){
    x=0;
    sampleCounterLine=0;
    y+=1;
   }
   x++;
  }

  //videoRenderArea.fillStyle="rgba(255,255,255,1)";
  //videoRenderArea.fillText(sampleIndex,0,10);

  window.requestAnimationFrame(RenderVideo);
 }
 function Reset(){
  videoRenderArea.clearRect(0, 0, videoCanvas.width, videoCanvas.height);
  index=0;
  sampleIndex=0;
  sampleTimer=0;
  sampleCounter=0;
  sampleCounterLine=0;
  timeStep=1/pixelClock;
  x=y=0;
  console.log(pixelClock);
 }

 RenderVideo();
});
```