Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Kókai Kristóf

INTERAKTÍV VIDEÓLEJÁTSZÁS

KONZULENS

Dr. Magdics Milán

BUDAPEST, 2021

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Kókai Kristóf**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2021. 12. 10.

...…….
Kókai Kristóf

Összefoglaló

A videók ma már a mindennapjaink részét képezik, azonban a jelenleg elérhető videótechnológiák szinte kizárólag csak a lejátszásra és annak minőségére fókuszálnak, elfelejtve, hogy a videókban jelentős kihasználatlan fejlődési potenciál rejtőzik más irányokban is. Egy ilyen lehetőség például azok interaktív funkcionalitással történő bővítése.

Ezalatt olyan opciókat lehet érteni, mint interaktív médiaelemek, hivatkozások, illetve reklámok megjelenítése, vagy olyan videók létrehozásának a lehetősége, ahol az egyes jelenetekben szereplő cselekmények kimenetelét több, előzetesen rögzített változat közül a néző határozhatja meg. Az ilyesfajta interaktív videólejátszás új távlatokat nyithat a videók felhasználásában az élet különböző területein, mint például a szórakoztatóiparban, lehetővé téve, hogy a médiafogyasztók egy filmben közölt történetnek ne csak passzív nézői legyenek, hanem akár aktív részesének, irányítójának is érezhessék magukat.

A felsorolt gondolatok megvalósítására már léteznek különféle megoldások, de ezek még mind elég kezdetlegesek és láthatóan inkább csak kísérleti jelleggel jöttek létre, pedig napjainkra az előfeltételek már rendelkezésünkre állnak, mind a hardverek, mind a videótömörítési- és továbbítási protokollok oldaláról.

Jelen dolgozat célja, hogy áttekintse az interaktív videók létrehozásához aktuálisan elérhető technológiai hátteret, bemutassa a főbb koncepciókat, majd végigvezessen azok tényleges felhasználásán is egy konkrét rendszer implementációján keresztül.

Abstract

Videos are now part of our everyday lives; however, the currently available video related technologies tend to focus almost exclusively on the playback and its quality, ignoring the fact that there is plenty of room for potential improvements in countless different directions as well. A perfect example could be a feature that allows us to add interactive functionality on top of normal video playback.

Interactive video playback could give us the ability to add interactive features such as interactive media elements, hyperlinks, or ads to our current and future videos, or even let us create videos where the viewers have the ability to pick between multiple pre-recorded continuations at the story's main decision points. These kinds of features could start a new chapter in the history of videos, opening up new possibilities in different areas of our lives, like the entertainment industry, where they could allow filmmakers to create movies that let the viewers feel like they are in direct control of the story, rather than just passive viewers.

There are already some existing solutions that implement the aforementioned ideas, but these are generally quite basic and often are created for experimental purposes only, even though the necessary technological prerequisites are already present both from the hardware side and the side of video compression and streaming protocols.

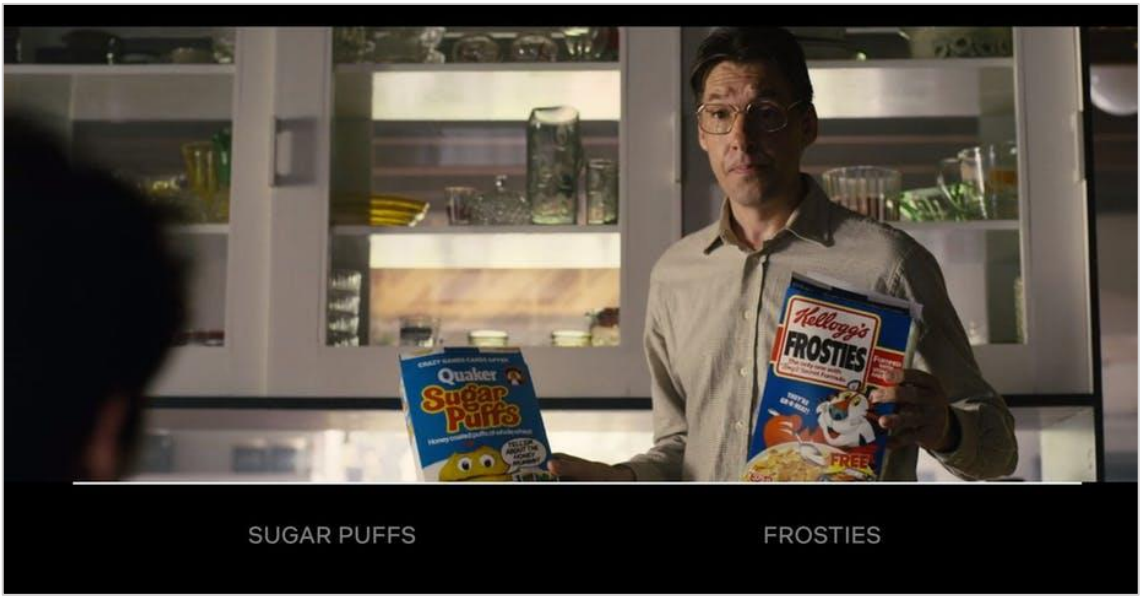
The purpose of the current document is to give an overview of the currently available technical background that is required for the creation of interactive videos, show the main concepts, then guide us through a specific technological implementation.

1 Bevezetés

Napjainkban a film- és mozgókép fogyasztás virágkorát éli, videók százezrei kerülnek megtekintésre nap, mint nap. Biztosan ezen dokumentum olvasói között is számos olyan ember akad, akik már találkoztak azzal a jelenséggel, hogy nem tetszett nekik egy-egy történet vége és ennek következményeként, felvetődött bennük a kérdés: „mi lett volna, ha…”.

Gondoljunk csak bele, mennyivel másabb élményt jelenthetne egy olyan film megnézése, amely lehetőséget ad nekünk arra, hogy a történet kulcskérdéseiben mi hozzuk meg a döntéseket a főszereplő helyett, ezáltal befolyásolva annak kimenetelét. Ezen gondolat mentén született meg, egyfajta válaszként, az interaktív videólejátszás ötlete; azonban, ha kicsit mélyebbre ásunk, még rengeteg más felhasználási lehetőséget is felfedezhetünk.

Az alábbiakban először motivációs jelleggel áttekintek néhány ilyen lehetőséget, majd ismertetem hol tart most a technológia és jelen körülmények között milyen opcióink léteznek egy interaktív videólejátszásra képes rendszer implementálására.



1. ábra Kép egy interaktív filmből[1]

[1] Forrás: [1]

1.1 Motiváció, felhasználási lehetőségek

Ahogy fentebb is említettem, az ötlet teljesen új távlatokat nyithat a filmiparban, de ezen felül más potenciális felhasználási lehetőségei is léteznek.

Egy ilyen, az eddigiektől eléggé eltérő terület a reklámipar. Gondoljunk csak bele, mennyire kényelmes lenne az, ha egy reklámban csak rákattintanánk egy termékre és rögtön megnyílna annak web vagy webshop oldala. Ráadásul, ez nem csak explicit reklámok esetén lehetne egy működő konstrukció, de például a korábban említett filmek esetén is, erre egy remek példa lehet az, ha egy filmben megtetszik nekünk valakinek a ruhája (vagy lényegében bármi más termék), akkor egy gombnyomásra meg tudhatnánk annak részleteit, például azt, hogy honnan tudnánk mi is beszerezni azt.

Mindezeken felül, akár kisebb, filmszerű játékok esetében is hasznos lehet egy ilyen technológia, elég csak belegondolni, hogy mennyire komplexek is tudnak lenni a modern játékmotorok, míg adott esetben ez egy sokkal egyszerűbb és rövidebb idő alatt megvalósítható alternatívát jelenthetne. Ezutóbbira jó példa lehet egy „Legyen Ön is Milliomos” típusú kvízjáték, ahol alapvetően videószerűen folyna a kvíz, akár csak a valóságban, de kihasználhatnánk az interaktivitást a válaszok megadása során.

Természetesen ezeken felül más példákat is ki lehetne találni, a lehetőségeknek kizárólag a képzelet szab határt, a fentiek inkább csak ízelítőként szolgáltak ahhoz, hogy lássuk ténylegesen mennyi potenciál is rejlik az ötletben.

2 Követelmények

Miután sikerült valamiféle képet kialakítanunk az interaktív videólejátszás ötletével kapcsolatban, ideje összegezni, hogy tulajdonképp milyen elvárásaink lehetnek egy azt realizáló ideális rendszerrel szemben:

- **Platformfüggetlenség:** Ahogy majd azt a következő fejezetből is látszódni fog, a jelenleg elérhető implementációk szinte kizárólag webes környezetben működőképesek, ellenben ideális esetben jó lenne, ha a kapott videók akár a webtől, mint platformtól függetlenül is lejátszhatók lennének.

- **Többféle alakzat támogatása:** a példákból is látszik, hogy az interakciós területek sokfélék lehetnek, ennek érdekében célszerű lenne, ha szabadformájú (pl. görbével határolt) felületek is megadhatók lennének az interakciókhoz.

- **Többféle interakciós esemény támogatása:** egy egyszerű és egyébként sok esetben elégséges megoldás lehet az, hogy az egyes interakciókra reagálva egyszerűen megváltoztatjuk az aktuálisan lejátszott videósávot, viszont pl. a korábbi reklámos példák esetén ez nem igazán jelentene megfelelő megoldást, hiszen itt teljesen más jellegű reakciót szeretnénk elérni (a termék weboldalának megnyitása), mint amit egy egyszerű ugrás a lejátszott videóban adhat. Természetesen egy ideális rendszer esetén ezen reakciók mindegyikét támogatni szeretnénk.

- **Minimális tárhely foglalás:** az ismertetett interakciók jellegéből adódik, hogy adott esetben azok tárolásához nagyobb mennyiségű adat lehet szükséges, amit, ha minden egyes releváns frame-ben el kell tárolni, akkor hatalmas videófájlokat kaphatunk, miközben a videókodekek használatának egyik legnagyobb előnye, hogy relatíve kezelhető tárigényű fájlokat eredményeznek. Azt mondhatjuk tehát, hogy az interakciók kezeléséhez szükséges extra adatok tárolásához szükséges lemezterület a lehető legkisebb kell, hogy legyen.

- **De(/en)kódolási hatékonyság:** a videók dekódolása erőforrásigényes, sokszor lassú folyamat tud lenni. Ezzel áll ellentétben az, hogy a felhasználók számára ezt a feldolgozási időt amennyire csak lehetséges szeretnénk elrejteni, hiszen lejátszás közben a legkisebb fennakadás is kényelmetlenséget jelent a nézőnek. Emellett, bár a megtekintési élményt nem befolyásolja, az enkódolás folyamata ennél jelentősen több időt is igénybe vehet, így ideális lenne, ha az is belátható időn belül végrehajtható lenne,

azaz az interaktivitási adatok kódolása nem okozhat jelenetős overhead-et ugyanazon videó normál esetben, bármilyen extra információ nélkül történő kódolási idejéhez képest.

- **Egyszerű tartalomkészítés:** ez inkább a videószerkesztő szoftverrel szembeni elvárás, de mindenképpen fontos szempont, hogy új interaktív videók létrehozása ne legyen túl komplex folyamat.

Véleményem szerint ezek egy ideális interaktív videólejátszásra képes rendszer legfőbb ismérvei.

A továbbiakban a fenti szempontok figyelembevételével próbálok a problémára megoldást találni.

3 Jelenleg elérhető technológiák

Némi keresést követően arra a következtetésre juthatunk, hogy az interaktív videólejátszás ötlete bár már korábban is felmerült[2], annak ténylegesen végfelhasználói fogyasztásra tervezett, működőképes implementációi csak az elmúlt egy-két évben kezdtek el megjelenni. Ebből kifolyólag, az ezekben megjelenő megoldások még eléggé kezdetlegesek, sokszor kompromisszumokkal telítettek és semmiképp sem mondható el, hogy létezne bármiféle szabvány ezek szabályozására.

Ilyen megoldások például a Netflix kísérleti jelleggel létrehozott interaktív filmjei, vagy olyan weboldalak, amik ezt szolgáltatásként ajánlják (általában némi díj ellenében). Ezekre tipikusan jellemző, hogy azok kifejezetten csak egy adott, szinte kizárólag webes környezetre lettek kialakítva, így például egy a számítógépünkre lementett videófájl esetében azok nem működnek, ráadásul az általuk kínált funkcionalitás is legtöbbször csak az alapvető igényeket fedi le.

Az általam felfedezett, legjobban kivitelezett interaktív videónak egyértelműen a [5] forrás alatt fellelhető videót nevezném.

3.1 Koncepciók

Munkám során több, a probléma megoldására mások által létrehozott alkalmazást is áttekintettem, ezekben az alapkoncepciók, ha nem is egyezőek, de gyakran elég hasonlóak voltak, viszont ugyanígy akadtak jelentős különbségek is az egyes szoftverek között, mind felhasználói, mind implementációs szinten, sőt, fellelhetők az interaktív videók szokványostól jelentősen eltérő interpretációi is, mint például a [6] „interaktív mozi” kísérletben, ahol az interaktivitás voltaképpen egy párnán keresztül valósult meg. Ebben a fejezetben a megismert legjellegzetesebb, legígéretesebb ötleteket igyekszem röviden áttekinteni.

Az az alapkoncepció, hogy adott egy videó és afelett van egy interakciós réteg az összes általam vizsgált rendszerben megjelent, amelyek az általam felvázolthoz hasonlóan értelmezték az interaktívitást, mint funkciót videókban.

[2] Például a [2] könyv, vagy a [3], [4] cikkek esetén.

Amiben talán a legnagyobb eltérések mutatkoznak az az elágazások kezelésének mikéntje, ami, mint majd látni fogjuk, a téma egyik legnehezebb és legtöbb problémát okozó kérdése. Itt a leggyakoribb megoldásnak időbeli ugrások használata tűnik, ez lényegében úgy működik, hogy van egyetlen egy (időben) hosszú videófájl és amikor a videó több irányba elágazik, akkor attól függően, hogy melyik irányba megy tovább a lejátszás, a videó eltérő időpillanatait megcélozva hajtunk végre ugrásokat. Egy ettől különböző gondolat, az, ha az idősávot ahelyett, hogy egy nagy egységként kezelnénk különböző részekkel, azt felbontjuk több kisebbre és a párhuzamos videósávokat cserélgetjük aszerint, hogy éppen melyik következik.

Az elágazásoknál felmerül még egy kérdés ez pedig az elágazó ágaknak a videóhoz képest történő struktúrálása. Ez alatt azt kell érteni, hogy milyen szerepet töltenek be a videóban az egyes leágazó ágak. Feltesszük-e, hogy van egy olyan történetszál, ami a videó fő, vagy „gerinc-vonulatának" tekinthető, vagy sem, és amennyiben igen, kötelező-e az egyes leágazásoknak valamikor visszatérniük erre a vonalra? Ezek azért lényeges kérdések, mert, ha mondjuk az elágazásokat egy gráffal szeretnénk leírni, akkor a fenti gondolatokra adott különböző válaszok függvényében több szignifikánsan eltérő gráfot kaphatunk.

További kulcskérdések, amikben a legtöbb lejátszó valamilyen szinten eltért, hogy milyen funkciókat és interakciós típusokat támogatnak, például találtam olyan, ahol konkrét form-okat is lehetett a videókba helyezni, majd beküldeni azokat, de olyan is, hogy az interakciók egyszerű időbeli ugrásokból álltak a videón belül.

3.2 Webes megoldások lehetőségei

A fenti koncepciók közül a weben egyetlen egy van, ami igazán dominánsnak nevezhető, de az olyannyira, hogy az implementációk közel 100%-ára igaz, ez pedig nem más, mint az egész videó egyetlen nagy idővonalként történő tárolása és az elágazások időbeli ugrásokként történő megvalósítása. Ennek elsősorban technológiai okai vannak, ugyanis weben a jelenlegi lehetőségeink mellett szinte lehetetlen több videósávot párhuzamosan kezelni, nagyon speciális egyedi megoldások nélkül.

Kicsit általánosabban ránézve a webre, mint platformra, elég jól látható, hogy számos olyan funkcióval rendelkezik, ami jelentősen megkönnyítheti az interaktív videók koncepciójának a platformba történő integrálását, gondoljunk csak a HTML5-ös videó

API-ra vagy a div elemekre. Többek között ennek tudható be, hogy a ma elérhető interaktív videótechnológiák legnagyobb része webre lett elkészítve.

3.3 Aktuális technológiák asztali környezetben

Asztali környezetben sokkal limitáltabbak a lehetőségek, annak okán, hogy a ma ott elérhető videólejátszási megoldások általában alacsonyszintű nyelvekkel és komplex algoritmusokkal dolgoznak, amiknek köszönhetően sokkal nagyobb energiabefektetéssel járna azok implementálása, miközben a kész megoldásunk várhatóan nem is feltétlen lenne platformfüggetlen, míg a weben ez tulajdonképpen adottnak tekinthető. Valószínűleg ezért nem is találtam egyetlen, még ma is aktívan fejlesztett asztali videólejátszó eszközt sem, ami képes lett volna interaktív videók lejátszására, pedig egyébként ugyanúgy minden adott lenne, mint a web esetén, csak az implementáció komplexitása az, ami lényegesen magasabb.

3.4 Személyes gondolatok

Abból kiindulva, hogy a videó feletti interakciós réteg lényegében minden általam vizsgált technológiában megjelent, és ha magamtól kéne nekiállnom a feladatnak én is hasonlóan kezdenék neki, kimondhatjuk, hogy ez az az alapötlet, ami köré utána minden további gondolatunkat építhetjük.

A videósávok és az egyetlen nagy idővonal ötlete lényegében ugyanazt csinálja, csak más formátumban. A videósávok használata nem rossz dolog, de ezzel lényegében elveszítjük (vagy legalábbis nagyon megbonyolítjuk) a lejátszási folyamat és az idők kijelzését, hiszen nem tudhatjuk biztosan hol tartunk és hogy az aktuális videósáv hogyan illeszkedik bele az egészbe. Az egy idővonalas megoldás használhatóbb reprezentációnak tűnhet, de itt ugyanúgy kezdeni kell majd valami az idő kijelzésével, mert kicsit furcsán nézne ki az, ha elindítanánk egy 5 órásnak írt filmet, aminek a nettó játékideje valójában csak maximum 2 óra.

Az elágazások strukturálása esetén én a teljesen szabad ugrások híve vagyok, ellenkező esetben nagyon lekorlátozhatjuk a videót megkreáló művész lehetőségeit.

Az ugrások és más különböző típusok támogatása pedig mindenképp fontos szempont szerintem, így igyekszem majd annyi interaktív elem típust hozzáadni az implementációmhoz, amennyi csak szükséges, hogy kényelmesen minden racionális lehetőséget meg tudjunk valósítani azok által.

4 Megvalósítási ötletek

Egy fajta ötletet már láthattunk a korábbiakban, ami teljes egészében webes technológiákra épített, az alábbiakban részletesebben is megvizsgálom azt, illetve más személyes ötleteimet, kiemelve a különböző megközelítések előnyeit és hátrányait.

4.1 HTML-alapú implementáció

Ezek a már eddig is említett webes megoldások, amik erőteljesen kihasználják a HTML DOM struktúráját és annak előnyeit. Bár a weben több megoldás is létezik, alapvetően azok lényegében azonos elv alapján készültek, amely a következőképpen fest: adott egy HTML5-ös videó elem, ebben kerül lejátszásra maga a videó. Ezek után az interakciós felületek (a képernyő síkjára merőleges Z-tengely mentén) efölött kerülnek elhelyezésre, amelyek megfelelő időpillanatokban történő megjelenítéséről, elrejtéséről, valamint az egyes kattintások megfelelő kezeléséről egy JavaScript parancsfájl gondoskodik előre megadott konfiguráció alapján.

A módszer viszonylag egyértelmű előnye, hogy a webes komponensek elég jó támogatást adnak a feladat megvalósítására, ezért egy ilyen lejátszó elkészítése sem lehetetlenül bonyolult, a videók megtekintéséhez pedig lényegében mindössze egy modernebb webböngészőre, illetve internet kapcsolatra van szükségünk. További előnyös tulajdonsága, hogy a több platformos támogatás a web által lényegében adott és egyetlen lejátszó elkészítése elég minden weboldalak megnyitására képes eszközre. Ráadásul a megoldás könnyen kiterjeszthető, valamint hordozható is, olyan értelemben, hogy ugyanazon JavaScript fájlokat felhasználva könnyedén hozzáadható más lejátszóhoz is az interaktivitás, mint extra funkcionalitás.

Hátránya, hogy a megközelítés láthatóan elég erősen webhez kötött, ami nem feltétlen rossz dolog, hiszen a webes tartalmak szinte mindenhol elérhetők, viszont mégiscsak jobb lenne egy valamivel általánosabb megoldást is találni a problémára. Továbbá, az interakciók adatainak tárolási formátuma még mindig egy nyitott kérdés, tekintve, hogy a webtől, mint platformtól nem kapunk semmiféle extra támogatást erre vonatkozóan, ellentétben mondjuk majd más, későbbi opciókkal. Ráadásul, amiatt, hogy a videókat webes tartalomként nézzük, azok letöltésére is szükség van, ami korlátozott sávszélességű és/vagy adathasználati keretű (pl. mobil) hálózatok esetén erőteljes

problémát jelenthet, sőt, akár kizáró tényező is lehet, tekintve, hogy az interaktív videók szokványos társaikhoz képest tartalmazhatnak alternatív videórészleteket is, ennek következtében jóval nagyobbak is lehetnek.

4.2 Videó lejátszása kész játékmotorban

Egy második, továbbra is többé-kevésbé egyszerűnek nevezhető megoldási ötlet lehet az, hogy fogjunk egy már kész játékmotort (pl. Unity), játsszuk le abban a videót, az interakciós felületeket pedig szimplán helyezzük el Z-irányban valahová a kamera és a videót tartalmazó sík közé.

A megközelítés legnagyobb előnye, hogy az előző módszerhez hasonló elven működik és lényegében roppant egyszerű, a realizációja nem igényel jelentősebb erőforrásokat, számos különböző típusú, valamint formájú interakciós területet is könnyedén tudunk vele kezelni, illetve már egy kész, ismert technológiára építünk, így szinte biztosak lehetünk benne, hogy amit elkészítünk, az működni is fog, legalábbis a választott játékmotor által támogatott kimeneti platformokon.

Sajnos, az előnyök itt ki is merülnek, a megoldásnak több, elég egyértelmű hátránya is létezik:

- Bár a legnépszerűbb nyilvánosan is elérhető játékmotorok által támogatott kimeneti platformok skálája elég széles, az mégiscsak limitált, egy-egy új platform támogatása pedig erősen idő- és erőforrásigényes lehet a motor fejlesztői részéről, ami számunkra azt jelenti, hogy várhatóan sokat kell várnunk addig, amíg egy esetleges újonnan megjelenő platformon is elérhetővé tudjuk tenni a megoldásunkkal készített videókat, így összességében elmondható, hogy a platformfüggetlenség csak meglehetősen szűk korlátok között garantálható.
- A következő probléma sem kevésbé jelenetős: mivel a videókat a játékmotorban szeretnénk futtatni, ezért azok lejátszásához az engine fájljaira is szükségünk lesz, ami azt eredményezi, hogy a szükséges tárterület drasztikus mértékben, a más megoldások által igényelt értékekhez viszonyítva többszörösére is megugorhat.
- További gond az, hogy lejátszáskor nemcsak magát a lejátszani kívánt videót kell feldolgozni, hanem azon felül még a játékmotort is futtatnunk kell, jelentősen megnövelve ezzel a szükséges CPU, GPU, valamint memória kapacitást és adott esetben a feldolgozási overhead-et.

4.3 Hagyományos videóformátumok használata

Az eddig ismertetett ötleteknek láthatóan megvoltak a maguk hátrányai, ezért célszerű lehet elgondolkodni azon, hogy nem tudunk-e esetleg egy új irányban jobbat is találni, például, ha a hagyományos videóformátumok már úgyis beváltak, miért nem azokból kiindulva próbálunk valamiféle megoldást találni? Lényegében ez a gondolat adja az alábbiakban ismertetett két módszer alapötletét.

Mivel a diplomaterv megvalósításához rendelkezésre álló idő véges, ezért ahelyett, hogy minden egyes videókodeket megvizsgálnék, kiválasztottam egy modernebbet, amellyel megpróbálom a feladatot megvalósítani. Esetemben ez nem más, mint a H.265-ös videókodek.

4.4 Interaktivitási információk tárolása feliratban

Amennyiben hagyományos videófájlokban gondolkodunk, eszünkbe juthat, hogy azokhoz alkalmanként már így is tárolunk kiegészítő információkat, ilyenek például a feliratok, amelyeket egyszerű feliratfájlokként tudunk a videókhoz csatolni. Adódik tehát a gondolat: miért ne használhatnánk ugyanezt a megközelítést az interaktivitási információk tárolásához, akár egy külön fájlformátum létrehozásával, akár a már létező technológiákra építve, egyszerűen csak a feliratokat használva valamilyen speciális prefix-szel?

A modern videóformátumok ráadásul már külön beépített felirat sávokat is tartalmaznak, így lényegében még külön fájlra se lenne szükség, hanem egyszerűen tárolhatnánk az interakciók adatait magában a videóban.

A módszer egyértelmű előnye, hogy az a korábbi videólejátszási technológiákra építve olyan szinten platformfüggetlennek mondható, hogy ahol már eddig is tudtunk videókat nézni, ott ennek is működnie kell, azzal a kitétellel, hogy a lejátszó szükséges, hogy rendelkezzen valamiféle támogatással a feliratok feldolgozására és megjelenítésére vonatkozóan.

További előnyként sorolható fel, hogy ez a megoldás lényegében az ideális interaktív videólejátszási rendszerrel kapcsolatos követelményeink egyikét sem zárja ki, azok legtöbbjének teljesülése kizárólag az adott implementáció hatékonyságától függ.

A módszernek két hátrányát tudnám megemlíteni: egyrészt azt, hogy egy eredetileg teljes egészében más célokra tervezett funkcionalitást (felirat) használunk fel

az implementáció során, ami semmiképpen sem egy szép megoldás és adott esetben akár különféle nem várt, kellemetlen mellékhatással is járthat, például akkor, ha a használt lejátszó nem támogatja az interaktív videólejátszás ezen formáját és standard feliratként próbálja megjeleníteni az interakciós adatokat. A megközelítés másik előnytelen tulajdonsága, hogy ahhoz, hogy interaktív módon játszhassunk le egy videót, már nem egy, hanem két fájlra is szükségünk lesz, ami kényelmetlenséget jelenthet a megtekintő számára.

4.5 Interaktivitási információk tárolása a videó stream-ben

Láthattuk, hogy a felirat nem feltétlen a legalkalmasabb hely az interaktivitási információk tárolására, így felmerülhet a gondolat, hogy nincs-e esetleg a videóknak más olyan része, ahol ezeket ideálisabb módon is elhelyezhetnénk, lehetőség szerint magába a fájlba belekódolva. A válasz az, hogy de igenis van[3], az alábbiakban egy ilyet fogunk megnézni.

4.5.1 SEI üzenetek

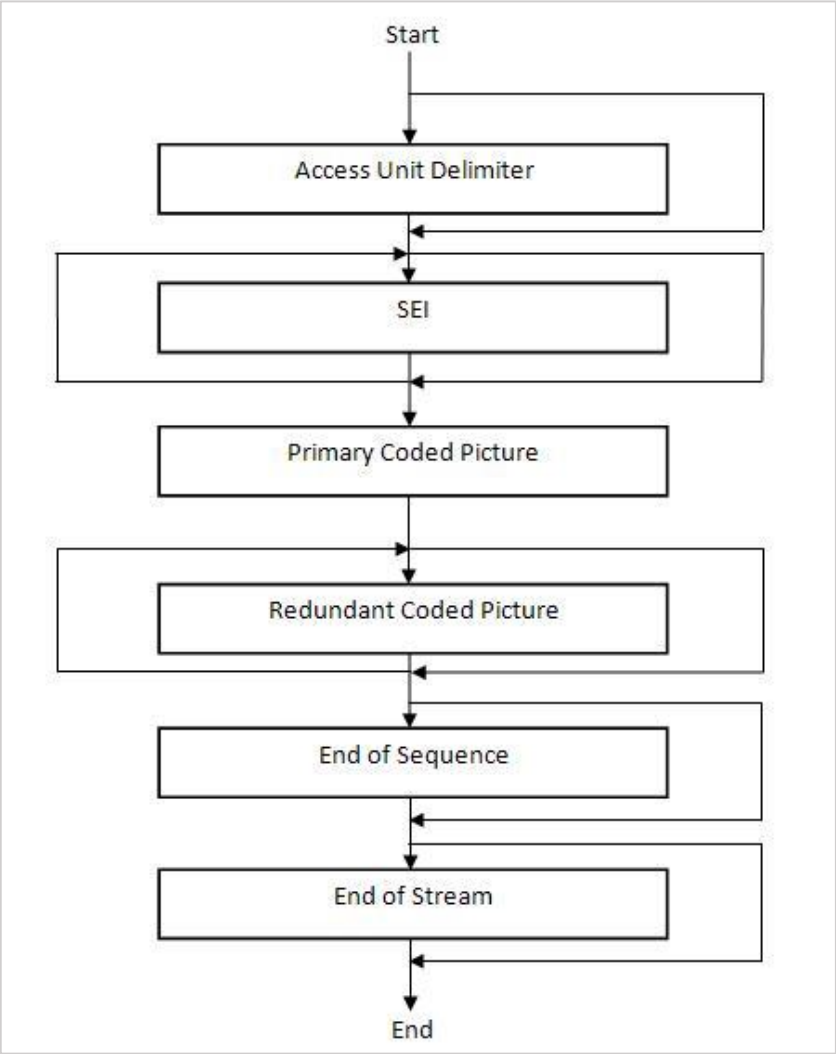
A SEI (Supplemental Enhancement Information) üzenetek olyan, a H.264 szabvány által definiált, majd később a H.265 szabvány [7] által is felhasznált kiegészítő információk, amelyek a videók lejátszása során lényegesek, azonban közvetlen képi (azaz luma = megvilágítási vagy chroma = színeket leíró) információkat nem tartalmaznak (pl. különböző szűrőkre, kamerára vonatkozó adatok stb.). A szabvány több előre megadott típusú üzenetet is specifikál, többek között olyat is (user_data_unregistered), amelyben bármilyen, a szabványban nem szereplő saját információt tárolhatunk. Az üzenetekre vonatkozó szabvány úgy lett kialakítva, hogy az bármikor bővíthető legyen további típusokkal, azonban szigorúan a szabványt követve minden olyan prefixű üzenet, ami még nem került bele a szabványba, kizárólag user_data_unregistered típusú üzenet formájában szerepelhet.

Fontos megjegyezni, hogy a SEI üzenetek egy-egy NAL Access Unithoz tartoznak, azok minden egyes NAL AU-hoz külön megadhatók.

[3] Legalábbis H.264 / H.265 esetén, sajnos a válasz erősen függ a választott videóformátumtól.

NAL (Network Abstraction Layer) Unit = a videó egy-egy frame-jének egy-egy kis szelete, a mérete tipikusan akkora, hogy egy átlagos hálózati csomagba beleférjen, ezzel téve jelentősen hatékonyabbá a videók hálózaton keresztül történő streamelését.

NAL Access Unit = több NAL Unitból felépülő egység, minden egyes dekódolt NAL AU-ból pontosan egy kép keletkezik, gyakorlatilag mondhatjuk azt is, hogy egy képkockát és a hozzátartozó kiegészítő információkat reprezentálja.



2. ábra Egy NAL Access Unit felépítése[4]

4.5.2 Interaktivitási információk tárolása SEI üzenetekben

Azt hiszem ezek után nem árulok el nagy titkot, ha azt mondom, hogy a SEI üzenetek ideális tárolóként szolgálnak az interaktivitási információk számára: minden

[4] Forrás: [8]

egyes frame-ben meg tudjuk adni, hogy ott éppen milyen interaktív területek láthatók/aktívak és mi történjen, ha azokkal interakcióba lépünk. Már csak annyi a kérdés, hogy pontosan milyen adatokat és milyen formátumban szeretnénk ezekben az üzenetekben tárolni. Ezekről és további, a megvalósítással kapcsolatos kérdésekről az implementációs részben (6.3 fejezet) fogok részletesebben is beszélni.

5 Megfontolandó kérdések

Mielőtt elkezdhetnénk implementációról beszélni, előbb akad még néhány kulcsfontosságú koncepcionális kérdés, amit szükséges tisztázni. Az alábbiakban ezeket fogom ismertetni és ahol van értelme, ott a saját megközelítésemet is részletezem.

5.1 Interakciók kezelése

Az első és legfontosabb kérdés maga a kiindulási koncepció, azaz pontosan mi is az az elképzelés, amire egyáltalán az implementáció során alapozhatunk.

Korábban bár már említettem, eddig különösebben nem részleteztem azt az ötletet, hogy az interakciókat képzeljük el, mint egy külön réteget a videó képe felett, amin különböző elemeket tudunk megjeleníteni és azokkal interakcióba lépni. Munkám során azt találtam, hogy lényegében ez a koncepció az, ami kivétel nélkül minden egyes lejátszóban, platformtól függetlenül működőképes lehet.

A fenti elképzelés kicsit részletesebben:

- A bemeneti adatok alapján az egyes interakciós elemek saját tartalommal, pozícióval, alakzattal, típussal és élettartammal (kezdő- és végidő/frame) rendelkeznek, függetlenül attól, hogy honnan érkeznek ezek az információk[5].
- A fenti adatok, megközelítéstől függően tárolódhatnak frame-enként (videó streamben tárolva), illetve globálisan is (külön fájlban), mindenesetre, ami biztos, hogy a lejátszónak ezeket valahol ki kell olvasnia, ahhoz, hogy döntést hozhasson arra vonatkozóan, hogy egy-egy elemet meg kell-e jeleníteni az adott *frame-ben/időpontban*[6]. Ha kicsit mélyebben belegondolunk, rájöhetünk, hogy a kettő között nincs érdemi különbség abban az értelemben, hogy az adott interaktív elem láthatósága

[5] Mármint úgy értve, hogy külön fájlban, feliratban vagy magában a videóban tárolódnak-e ezek az adatok.

[6] Ebben és a következő az alfejezetben a továbbiakban csak frame-ként hivatkozok a videó egy-egy időpillanatára. Ezzel kapcsolatban részletesebben információ a 5.3 szakaszban olvasható.

úgyis a konkrét frame megjelenítésekor kerül eldöntésre, így ebben a fejezetben nem is fogok különbséget tenni a két megközelítés között a továbbiakban.

- A fenti pontok is implikálták, de eddig nem lett explicit kimondva: egy elem a hozzárendelt kezdő frame-ben jelenik meg a neki megadott pozícióban, a megfelelő tartalommal és alakban, majd az utolsóként megjelölt képkockáját követően eltűnik. Ennek ellenőrzése minden egyes frame-ben, minden egyes elemhez megtörténik.
- Az egyes interakciókat viselkedésük alapján kategorizálom (pl. hiperhivatkozás, elágazás, …), ez az, amire az interakció típusaként hivatkozom. Ennek részletekbe menőbb kifejtése a 5.2 fejezetben található.
- Az interaktív reakciós folyamat az egyes elemekre történő kattintás esemény bekövetkeztével indul el.

5.2 Interakciók formái

Az eddigiek során már párszor említésre került, hogy az egyes interakciós elemeknek különböző típusai fordulhatnak elő, néhol esetleg még pár példa is megjelent ezekre. Ebben a fejezetben alaposabban is leírom, hogy az egyes interakciókat milyen kategóriákba sorolhatjuk viselkedésük alapján.

5.2.1 Egyszerű tartalom

A legelemibb és legegyszerűbb elem, amely alap esetben gyakorlatilag nem is interaktív, csak szimplán a neki megadott tartalmat jeleníti meg. Annak, hogy mégis rákerült a listára két oka van: egyrészt a többi elem mellett a teljességhez szükséges és bizonyos más interaktív elemekkel kombinálva különféle kiegészítő funkciókat láthat el; másrészt, bár ez platformtól függ, de például webes környezetben interaktívvá tehető, ha a tartalmát nemcsak szöveges formában, de HTML-ben is definiálhatjuk. Ezutóbbi képesség birtokában akár olyan komplex dolgokat is el tudunk készíteni, mint egy kitölthető form a videóban.

5.2.2 Ugrás

Az előző elemhez viszonyítva, ez már egy fokkal komplexebb. Mint bármely más típusú komponens, ez is a neki megadott tartalmat jeleníti meg, de amennyiben arra a tartalomra rákattintunk, akkor egy a videón belüli ugrást hajt végre a neki interakciós célként megadott frame-re, attól a képkockától folytatva a lejátszást.

5.2.3 Elágazás

Nagy valószínűséggel a legfontosabb és egyben legtöbb problémával járó komponens. Végeredményben elmondható, hogy ez az elem alkotja az egész interaktív videólejátszás motorját. Funkcionalitás szempontjából hasonlít az ugrásra, azonban van egy hatalmas különbség: egy elágazás blokkolja a videó további lejátszását. Ez annak okán lényeges, hogy egy történetbeli főbb választási pontok ezzel az elemmel valósíthatók meg, hiszen amikor döntési helyzetbe kerülünk, a lejátszás addig áll, amíg a néző nem nyom rá valamelyik folytatási opcióra.

Az elágazások megvalósításához szükséges legnagyobb kérdés az, hogy az egyes ugrási célként meghatározott alternatív videósávok hol helyezkedjenek és hogyan érhetjük majd el azokat. Erre alapvetően két megközelítést találtam. Tárolhatjuk azokat egy videósávon, azonban ez furcsa mellékhatásokkal járthat és különböző, közel sem triviális problémák merülhetnek fel például a seeker-rel vagy a lejátszási idővel kapcsolatban[7]. A másik megoldás az lehet, hogy minden egyes alternatív videórészletet külön videósávon[8] tárolunk és megjelenítéskor váltogatunk azok között annak függvényében, hogy a néző milyen ugrásokat hajt végre. Sajnálatos módon ezutóbbi ötletnek is számos sarkalatos pontja akad, talán még több is, mint az előzőnek, ezek közül néhány:

- Több videósáv tárolását és azok között ilyen módon történő váltásokat támogató videóformátumok száma erősen limitált.

[7] Részletesebben lásd.: 5.6 fejezet

[8] Vagy akár külön fájlban is, de ez további gondokkal járhat hordozhatóság szempontjából. Gondoljunk csak bele, ha egy interaktív filmben 100 alternatív részlet van, akkor a felhasználónak 100 videófájlt kéne hordoznia, ha pl. át szeretné másolni a videót máshová.

- Bizonyos platformokon (például weben) ez a koncepció elég nehézkesen kivitelezhető és jelentős többletmunkával jár.
- Az egyes alternatív fájlok/sávok betöltését úgy kell megoldani, hogy a váltások során a lejátszás mindvégig folyamatos maradjon.

Mindent összevetve a saját implementációmban az első megközelítést alkalmaztam, tekintve, hogy az egy általánosabb és talán kevesebb problémával járó megoldást kínál, abból kiindulva, hogy bár a felmerülő pl. lejátszás idejével kapcsolatos komplikációk közel sem egyértelműek, azok véleményem szerint még mindig megoldhatóbbak, mint a másik megoldásnál felsorolt gondok.

Az elágazásokhoz kapcsolódó akad még egy nagy kérdés, aminek a relevanciája, illetve több kulcsfontosságú részlete ugyan csak a későbbiekben fog tisztává válni, azt mégis érdemes itt tisztázni, mivel a továbbiakban az erre adott válaszomat már adottként szeretném kezelni, annak érdekében, hogy a későbbi fejezetek könnyeben érthetőek és követhetőek legyenek. Ez a kérdés nem más, mint az egy elágazási ponthoz tartozó elemek típusainak és együttes viselkedésének szabályai, erre szintén két fő gondolatom van:

- Egy elágazási pontban egyetlen elágazási elem van, amely lényegében csak adminisztratív funkciókat lát el, azáltal, hogy jelöli az elágazást, az egyes döntési irányokba pedig egy-egy szimpla ugrás elem segítségével mehetünk tovább. Ennek megvan az az előnye, hogy az elágazások nagyon könnyen követhetők az implementációban, azonban az a hátránya is, hogy egy plusz elemre van szükségünk a megvalósításhoz, ami lényegében semmilyen tényleges interaktív funkcióval nem rendelkezik és ráadásul az elágazásban, vagy loopolt elágazás (erről részletesebb ismertető a következő fejezetben olvasható) esetén a teljes loop ideje alatt nem használhatunk ugró komponenseket más célokra, például időben visszafelé történő ugrásra.
- Minden egyes irányba tovább vezető interakciós elem egy-egy elágazás és nincs adminisztratív komponens. Előnye, hogy nincs extra elem, az egyes komponensek típusa intuitív módon megadható. Hátránya, hogy az egyes interaktív elemek élettartamánál megadott időértékeknek minden elem esetében logikusan meg kell egyezniük, különben az adott lejátszó

implementációjától függően különféle anomáliák fordulhatnak elő az elágazások megjelenítésekor. Ez különösen loopolt elágazások esetén fog meghatározó kritériummá válni.

Az általam megvalósított rendszerbe a második megoldást építettem be, ennek oka egyrészt az, hogy így nem korlátozódik az ugrások használata, másrészt majd a szerkesztőnél látható lesz, hogy ez a videók készítői számára is egy jelentős könnyebbséget jelentő megközelítés.

5.2.3.1 Videó loopolása elágazások esetén

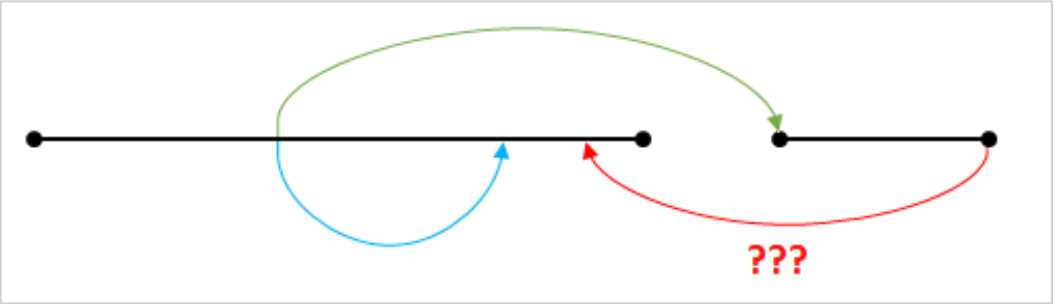
Az elágazások egy speciális típusa a loopolt elágazás. Itt arról van szó, hogy amíg a felhasználó döntésére várakozunk, ahelyett, hogy a lejátszás blokkolódna, a videó utolsó X másodpercét játsszuk le újra és újra. Megfelelő művészi munkával könnyedén elérhető, hogy ezáltal a videó lejátszása folyamatosnak tűnjön az elágazások közben, növelve ezzel a felhasználói élményt.

Ez az extra funkcionalitás láthatóan egy új interakciós típust kíván annak érdekében, hogy az könnyen és hatékonyan kezelhető legyen az implementációban. Innentől az egyetlen még nyitott kérdés az, hogy pontosan hol, illetve hogyan érdemes tárolni azt a videó részletet, amit ismételni szeretnénk. Ehhez felvehetnénk teljesen új adatokat is, azonban egy sokkal kézenfekvőbb és egyébként jobb teljesítményű megoldás lehet az, ha ehelyett felhasználjuk a már egyébként is tárolt információkat és az elem kezdő, valamint utolsó frame-je fogja meghatározni a loopolandó szakaszt. Ez annak következtében tehető meg, hogy az elágazás egyébként egy álló dolog, ami lényegében egyetlen időpillanatig tart, így a kezdő és vég időpontok erre a célra szabadon felhasználhatók maradnak.

5.2.3.2 Automatikus ugrások

Az eddig megismert információk tudatában már nincs különösebb technikai akadálya az elágazások elkészítésének, viszont, ha azok valós felhasználásába is belegondolunk, felmerül egy komoly praktikussági probléma, az alábbiakban ennek részleteit ismertetem és egyben egy megoldást is kínálok rá, az automatikus ugrások formájában.

A probléma egy esetét az alábbi idővonal szemlélteti:



3. ábra Ugrási probléma demonstrációja

A képen a következőket látjuk:

- A videó kezdete az első fekete intervallum bal oldala, a vége (jelenleg) minden esetben ugyanezen vonal jobb széle.
- A pont ahonnét a kék és zöld nyilak kiindulnak egy elágazási pont, a kék és zöld elágazások az egyes nyilak végénél található időpillanatokra ugranak.
- A jobb oldali fekete intervallum egy alternatív videórészlet az elágazási pont és a piros nyíl vége közötti szakasz helyett, ha a néző a zöld elágazási irányt választja.
- Ezek után látható, az ábrán egyébként kérdőjelekkel jelölt probléma: ha a jobb oldali sávról vissza (vagy akár tovább) szeretnénk ugrani, akkor nekünk ott egy ugrás szükséges, de ha ezt a korábbi ugrás típussal szeretnénk implementálni, akkor ott egy felhasználói interakció lenne szükséges, ami valójában teljesen szükségtelen, mert pontosan tudjuk, hogy hol, melyik frame-mel fog folytatódni a lejátszás és mindezzel összességében csak felesleges kényelmetlenséget okozunk a felhasználónak.

Erre a nehézségre a megoldást egyfajta automatikus ugrás formájában képzeltem el: lényegében ez egy tartalom nélküli, egyetlen frame-ig tartó ugrás elem, amelynek kizárólagos célja, hogy jelölje hol fog folytatódni a lejátszás, amikor elértünk egy alternatív intervallum végére.

5.2.4 Hiperhivatkozás

Ez egy elég egyszerű elem: lényegében egy szimpla tartalmi elem, amire rákattintva megnyitja az adatai közt hivatkozott weboldalt. Értelemszerűen, a webtől eltérő platformon futó lejátszó esetén, ahhoz, hogy ez az elem működőképes lehessen, szükség van az azt futtató eszközön egy webböngészőre a hivatkozások megnyitása céljából.

5.2.5 Interaktív kártya

Ez a komponens körülbelül azt takarja, amire első ránézésre gondolhatnánk: egy olyan nyitható, illetve zárható kártya, ami a tartalom felett lebeg és van valamilyen tartalma, amivel más interaktív elemekhez hasonlóan interakcióba léphetünk. A kártya megjelenítése, avagy kinyitása történhet automatikusan, vagy egy másik elemmel interakcióba lépve, lényegében ez az, ami a többi interaktív komponenstől megkülönbözteti ezt az elemet. A felhasználása elsősorban reklámok esetén lenne elképzelhető, a motivációs fejezetet idézve, például amikor egy termékre rákattintva megjelennek annak részletei, vagy mondjuk filmek esetén, ahol a videó egyes részleteivel interakcióba lépve további információkat tudhatunk meg az adott dologról, valamint az elmesélt történet hátteréről. Az elem megvalósításának komplexitása erősen változhat, a tartalmazó lejátszó és a megvalósítani kívánt animációk függvényében, az általam elkészített webes lejátszó esetében például ez valahová a lista közepére esett, de teszem azt egy asztali videó megjelenítő alkalmazás esetén a probléma sokkal bonyolultabbá is válhat.

5.2.6 Cserélhető tartalom külső adatforással

Felmerülhet az igény, példának okáért a korábban bemutatott stream-alapú adattárolást megvalósító konstrukció esetén, hogy egy-egy interaktív komponens tartalma könnyedén lecserélhető legyen a teljes videó újra renderelése nélkül is, erre jó megoldást nyújthat egy új elem bevezetése, amelynek a tartalma a videón kívülről származik, ezáltal az cserélhető marad, anélkül, hogy a videószerkesztő szoftverhez akár csak egy pillanatra is hozzá kéne nyúljunk. Egy ilyen komponens például reklámok esetén lehetne felettébb praktikus, hiszen így a megjelenített reklám tartalma könnyedén leváltható lenne, mondjuk abban az esetben, ha a videót megosztó cég szerződése lejár a saját termékét reklámozni kívánó céggel. A felhasználási lehetőségeket végig gondolva, a komponens tartalma egyaránt lehet kép, egy másik videó, vagy csak egy egyszerű szöveges üzenet is.

Ezek alapján a megvalósításhoz célszerű egy új elem-specifikus paraméter felvétele, amely azt határozza meg, hogy a külső tartalommal rendelkező komponens mely másik elemhez hasonlóan viselkedjen (pl. hiperhivatkozás, ugrás, egyszerű elem, …). Mindebből az is jól látható, hogy itt az implementáció valamivel komplexebb, mint a legtöbb eddig megismert komponens esetén, amellett, hogy egyébként különféle biztonsági aggályokat is felvethet a külső tartalmak megbízhatóságára vonatkozóan. Ezutóbbi kérdésben a legjobb, amit tehetünk azt hiszem az, hogy az ilyen jellegű tartalmak kétséges mivoltáról a felhasználókat is értesítjük.

5.3 Frame vs. idő

Alapvetően, amikor azt szeretnénk valahogy követni, hogy időben hol vagyunk egy videóban, jellemzően azt az aktuális frame azonosításával, például annak sorszámával[9] szoktuk megtenni, aszerint számozva ahogy az egyes képkockák sorban jönnek a videó lejátszása esetén. Ezzel ellentétben, bizonyos környezetekben ez egy konkrét időpont megadásával történik, tipikusan töredék másodpercekig visszamenően, mint mondjuk a HTML5 által biztosított JavaScript Video API-ban, ahol jelenleg nincs is lehetőség arra[10], hogy az aktuális frame-et megkapjuk[11].

Mindent mérlegelve az alábbi döntésre jutottam: mivel a legtöbb helyen az tekinthető a standard megközelítésnek, így ahol lehetséges frame azonosítókat használok, ezalatt elsősorban az asztali környezetet kell érteni stream-alapú adattárolással, ahol az információk megőrzése egyébként is a frame-ekben történik, azon platformokon, ahol pedig ez nem lehetséges, ott jobb híján időpontokkal dolgoztam.

Ennek következményeként, sajnálatos módon, egy kisebb inkonzisztenciát hoztam be a rendszerbe, – legalábbis átmenetileg, amíg webes környezetben is mélyebb hozzáférést kaphatunk a videó en- és dekódolási folyamathoz – hiszen így a

---

[9] Erre egy konkrét gyakorlati példa a 6.3.1 fejezetben olvasható.

[10] Különböző kerülő megoldások viszont léteznek, de ezek tipikusan mind arra építenek, hogy ismerjük az éppen megjelenített videó frame rátáját, ami jelen esetben sajnos nem lehet követelmény. Például egy ilyen: [9]

[11] A W3C éppen a napokban adta ki WebCodecs nevű API vázlatát, ami egy olyan interfészt definiál, amivel közvetlenül is elérhetjük a böngésző videó en-, illetve dekóderét, ezáltal többek között már a videó frame-ekhez is hozzáférést biztosít. Forrás: [10]

videószerkesztő szoftvernek különböző outputok előállítására is képesnek kell lennie a kimeneti platform függvényében.

Megjegyzés: Ez egyébként ettől függetlenül is szükséges, amiatt, hogy weben a stream-be kódolt információkat (SEI üzenetek) sem érjük el, így az interaktív bemeneti adatokat kénytelenek vagyunk egy külön fájlban tárolni.

5.4 Transzformációk (animációk)

A videó céljától függően szükséges lehet az interaktív elemek animálása, ezt azoknak az idő előrehaladtával történő transzformálásával érhetjük el, így fontos kérdés lehet, hogy milyen lehetőségek vannak ennek megvalósítására és a transzformációk végrehajtásához szükséges adatok tárolására.

5.4.1 Egyszerű

Az első és legegyszerűbb ötletünk azt lehet, hogy a videó úgyis kétdimenziós, így szimplán eltárolhatjuk az elem aktuális állapotát (pozíció, elem középpontja körüli forgatás szöge, az elem méretére vonatkozó skálatényező) minden frame-ben, amíg a komponens látható, így szimplán annyi a teendőnk, hogy mindig az aktuális állapot beállítások mellett rajzoljuk ki az elemet.

A megoldás egyértelmű előnye, hogy rendkívül egyszerű, kevesebb adatot tárol, mint a következő opció, valamivel gyorsabb is és az esetek meglepően nagy részében lényegében elégségesnek is bizonyul. Akadnak olyan eshetőségek is, amikor ezutóbbi nem igaz, azokban az esetekben lesz segítségünkre a második ötlet.

5.4.2 Mátrix alapú

Néha az egyszerű megközelítés nem feltétlen elégséges, gondoljunk csak a 360°-os videókra, vagy arra az esetre, ha mondjuk valamilyen háromdimenziós animációt, vagy akár csak több tengely menti forgatást szeretnénk megvalósítani. Előbbi esetben, bár nem lenne egyszerű, de a feladat még megoldható is lenne csak az eddig megismert gondolatok alapján, hiszen a 360°-os fokos videó lényegében csak arról szól, hogy kiterjesztjük a field of view-t egy sima videóhoz képest, az utóbbi helyzetre viszont ez már semmiképp sem igaz, itt már valami másra is szükségünk lesz. Ez a valami más nem más, mint egy klasszikus transzformációs mátrix, ami mindezt lehetővé teszi számunkra.

A koncepció a következő: az egyes animált elemekhez minden frame-ben tárolunk egy-egy transzformációs mátrixot, amivel a megjelenítés előtt mindig eltranszformáljuk az adott elemet. A videószerkesztő szoftverek tipikusan keyframe alapú animációkat használnak, így az egyes mátrixok a keyframe-ekben a megadott transzformációk alapján beállíthatók, míg a keyframe-ek közötti frame-ekben interpoláció segítségével legenerálhatók. A módszer egyetlen tényleges hátránya az implementáció megnövekedett bonyolultsága, illetve talán az, hogy annak van némi teljesítménybeli költsége, hogy a CPU-n kell mátrixszorzást végrehajtanunk, de ez a videó dekódolásához képest elenyésző overhead-et eredményez. Ugyanígy felmerülhet az is problémaként, hogy így megnövekedhet a szükséges tárolási kapacitás, azonban, ha belegondolunk ez 4x4-es mátrixok esetén, 4 bájtos szám típusok mellett mátrixonként 64 bájtot jelent, ami mondjuk egy fél millió frame-en keresztül látható objektum esetén, 60 fps-sel történő lejátszás mellett, tehát egy kb. 3 órás videórészlet esetén is csak kb. 32 MB többletet eredményez tömörítés előtt (!), így, ha a videó egészét nézzük, akkor ez még egy elfogadható mennyiség, tekintve, hogy a fenti egy viszonylag extrém eset volt és bár a növekedés egyértelműen megfigyelhető, az mégsem olyan hatalmas mértékű ahhoz a tárterülethez képest, amit a videó többi része elfoglal.

A saját megoldásomban a webes lejátszó esetén az egyszerűbb megközelítést alkalmaztam, míg az elkészített enkóder/dekóder alkalmazás esetén már transzformációs mátrixokkal számoltam, ezekről részletesebben az implementációs részben fogok beszélni.

5.5 Pufferelés

Ha videólejátszásról beszélünk, akkor elengedhetetlen, hogy az annyira folyamatosan történjen, amennyire csak lehetséges, ezt normál videólejátszás esetén pufferelléssel szokták biztosítani. Ez egy olyan technika, ahol a videó soron következő részei előre betöltésre kerülnek, így amikor azok lejátszására kerülne a sor, a megjelenítendő képek már a memóriában rendelkezésünkre állnak.

Pufferelés alatt gyakran két különböző dolgot szokás érteni: egyrészt a képek pufferelését lejátszáskor, hogy azok megfelelő sebességgel megjeleníthetők legyenek, másrészt webes videólejátszás esetén a hálózati pufferelést, ahol még a megtekintést megelőzően a videófájl kisebb szeletekre (chunk) kerül „felvágásra” és a releváns darabjai a videó nézése közben előre töltődnek le a néző számítógépére.

A következő részekben arról fogok beszélni, hogy az interaktivitás bevezetése hogyan érinti ezeket a funkciókat és az egyes esetekben mit lehet tenni a videólejátszás folyamatosságának biztosítása végett.

5.5.1 Képi puffer

A videó képének pufferelése során a videó soron következő képkockái egy memóriabéli puffer tárba kerülnek, ahonnét majd kiolvasásra kerülnek, amikor a lejátszásukra kerül a sor. Ennek a tartalma videó formátumonként és szabványonként változó, hogy hogyan kerül eldöntésre, például az általam vizsgált H.265 esetén a DPB (Decoded Picture Buffer) tartalmát az enkóder szabhatja meg frame-ről frame-re[12], de ez nincs mindig így.

Az interaktivitás bevezetése a kép puffer kezelésében elkerülhetetlen változásokat okoz, konkrétan a problémát az elágazások (és ugrások) használata jelenti, mivel itt a videó lejátszásának irányát a felhasználó döntheti el, így előre nem tudhatjuk, hogy pontosan melyek azok a frame-ek, amiket pufferelni kéne. Ez a bonyodalom a legtöbb népszerű formátum esetén orvosolható, viszont, ahogy már említettem az egyes formátumok között vannak eltérések arra vonatkozóan, hogy hogyan szabályozzák a képi puffer tartalmát, így részletesebb megoldást csak az általam használt H.265 esetében tudok adni.

H.265 esetén a probléma megoldásának kulcsát annak a fejezet elején ismertetett tulajdonsága jelenti, nevezetesen, hogy minden egyes frame-ben az enkóder meghatározhatja a DPB tartalmát. Ezt kihasználva, már nem is tűnik olyan komplexnek a feladat, lényegében arról van szó, hogy elágazások, illetve ugrások esetén csak minden egyes irányt, amerre a lejátszás folytatódhat pufferelünk. Ennek a pontos folyamatát és további részleteit a 6.3.1.2 fejezetben ismertetem az enkóder más implementációs kérdéseivel együtt.

5.5.2 Hálózati puffer

A hálózati pufferelés megvalósítása már a lejátszó feladata, de itt is számolnunk kell az interaktivitás okozta bizonytalansággal. Alapvetően H.265 esetén a fájlok strukturálása segíti mindezt, hiszen azok NAL Unit-onként szét vannak bontva, így a képi

[12] Részletesebben lásd.: 6.3.1.2 fejezet

puffer alapján az adott lejátszó már tudhatja, hogy mely következő darabokat kell előre betölteni, ezzel lényegében enkóder-dekóder oldalról meg is van oldva a kérdés, a továbbiak már a lejátszóra vannak bízva.

Ugyanakkor, munkám nemcsak egy enkóder, illetve dekóder szoftverrel, de egy webes lejátszóval is foglalkoztam, ahol már ténylegesen is megjelenik a probléma. Sajnálatos módon, ahogy már korábban is említettem a weben általam használt API nem nyújt hozzáférést a videóstream-ben tárolt extra adatokhoz, így valami más megoldást kellett találnom erre a problémára.

Az ötlet a következő: egy weboldalon belül akár több videó elemet is létrehozhatunk, amikor ugráshoz vagy elágazáshoz érkezünk, minden egyes ugrási irányhoz hozzunk létre egy-egy ilyen nem látható HTML node-ot a háttérben, adjuk hozzá azokat a DOM-hoz, a videók forrása legyen ugyanaz a fájl, mint a megjelenített videóé és állítsuk be ezek mindegyikét azon időpontok egyikére, ahol a lejátszás folytatódhat. Ezzel a böngészőt rákényszerítjük arra, hogy előre letöltse a pufferelendő szeleteket. Értelemszerűen, amikor kijutunk egy ugrásból vagy elágazásból, a létrehozott videóelemeket törléséről gondoskodnunk kell, ellenkező esetben memória szivárgás fog megjelenni a programunkban.

A megközelítés egyértelmű hátránya, hogy adott esetben jelentősen fokozhatja a hálózati forgalmat, valamint az alkalmazás memória igényét, de ennél jobb megoldást egyelőre nem találtam.

5.6 Seeker

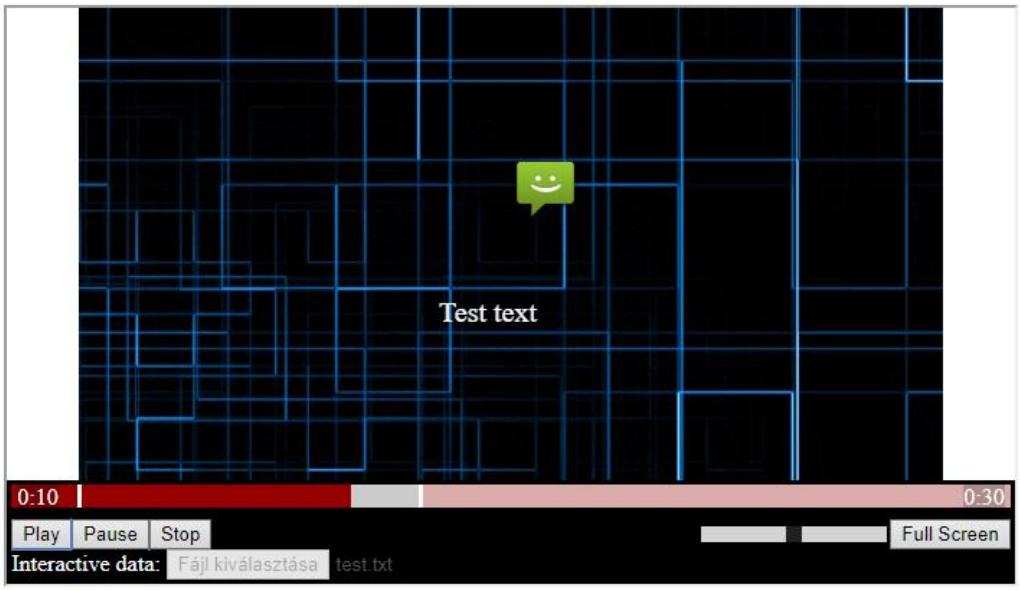
Az interaktivitás bevezetése a videó seeker funkciójában (idővonal, tipikusan a videólejátszók alján, amely segítségével a videót előre-, illetve visszatekerhetjük) is szükségessé teszi bizonyos változtatások bevezetését. Gondoljunk csak bele, alap esetben a videó lejátszása leírható egy egyenes vonalon, azonban ebben az esetben a különböző elágazások és ugrások megjelenése gyakorlatilag egy az egyben elrontja ezt a funkciót, mivel itt a videó lejátszásának folyamata inkább már egy szerteágazó gráffal, mint egy szakasszal írható le megfelelően, ezzel megfosztva minket a seeker egyszerűségének kényelmétől.

Az alábbiakban olyan megoldásokat vizsgálok meg, amelyekkel kiküszöbölhető ez a probléma, úgy, hogy közben a seeker használatának komfortossága is a lehető legkisebb mértékben sérüljön.

5.6.1 Bookmarkok

A kiindulási ötlet a következő: tartsuk meg a már jól megszokott egyenes idővonalat és azon, egyfajta könyvjelzőként jelöljük meg az egyes elágazások helyét, azokra kattintva pedig ugorjunk az elágazások elejére. Ezen felül fontos, hogy elágazásokat ne tudjunk átugrani, ennek meggátolására bevezettem azt a szabályt, hogy az időben következő bookmark utáni időpontra nem engedjük a felhasználót navigálni, amennyiben mégis oda szeretné tekerni a videót, helyette a legközelebbi következő elágazásra ugrunk.

Arra, hogy ez hogyan is néz ki a gyakorlatban, egy példát az alábbi képen láthatunk:



4. ábra Bookmarkok megjelenítése a seekeren

A seeker bar a videó alján található gombok feletti sáv. A képen a sötét vörös terület jelzi a már lejátszott részeket, a szürke az, ami még a következő elágazásig hátra van, a fehér csíkok a bookmarkok, avagy elágazások, a világos piros pedig a következő elágazás utáni, még megtekintetlen részeket jelzi, erre kattintva a fentiek szerint a videó lejátszása a szürke sáv jobb szélén található elágazáshoz ugrik.

Ezzel a megoldással lényegében csak kiegészítettem magát a seekert, annak alapfunkcionalitása megmaradt, de közben már az interaktív funkciókhoz is támogatást biztosít. Az egyetlen dolog, ami ebből igazán hiányzik az az, hogy azt is láthassuk, hogy az egyes elágazásokból milyen irányokban haladhatunk tovább és adott esetben könnyedén tudjunk navigálni azok között. A következő alfejezet ezt próbálja áthidalni további kiegészítések bevezetésével.

5.6.2 Seeker-gráf

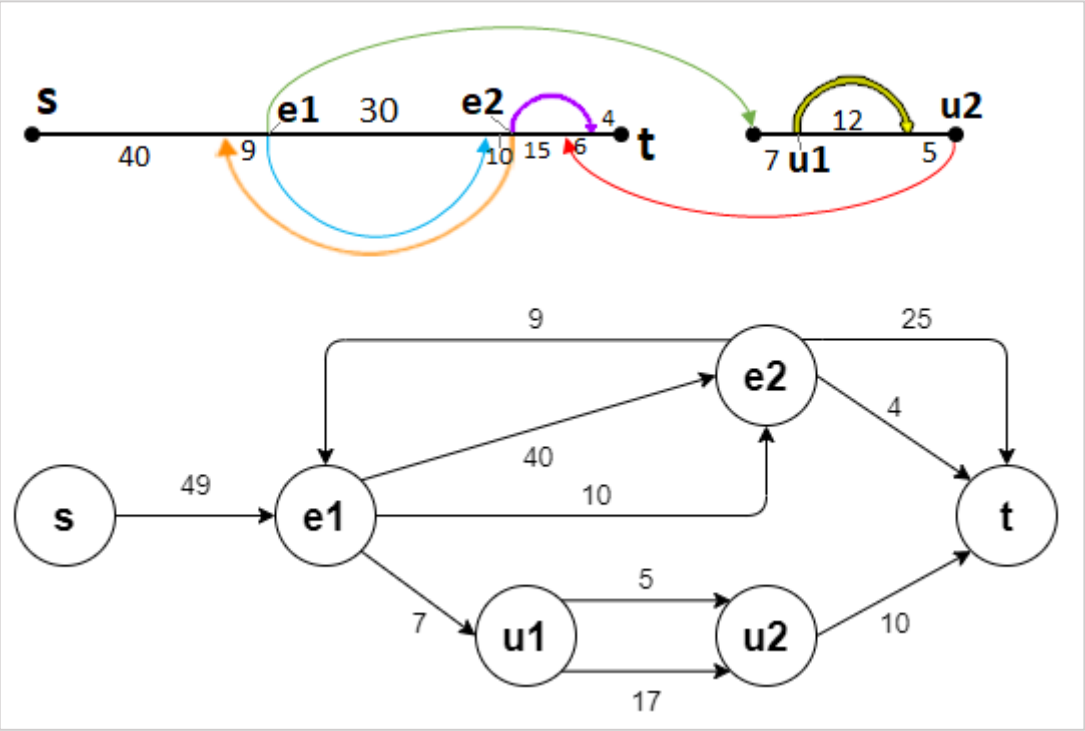
Megjegyzés: az ebben az alfejezetben szereplő gondolatokat csak elméleti szinten vizsgáltam, azok tényleges kipróbálására sajnos már nem került sor, és bár ezen ideák egy részének megvalósítása semmiképp sem triviális, semmilyen elvi akadályát nem látom azok implementálásának.

Az imént bemutatott bookmarkos gondolat egy elfogadható megoldást nyújtott interaktív videólejátszás esetén a seeker kezelésére, ugyanakkor abból hiányzik még a teljes szerteágazó videófolyam, mint egy gráf navigációs lehetősége, amivel az még kényelmesebbé tehető lenne.

Ami egyértelmű, hogy ezt a gráfot nem tehetjük a seeker aktuális helyére, hiszen az vagy túl nagy kiterjedésű helyet foglalna el, vagy egy rémálom lenne a navigálása, tehát valami más elgondolásra van szükség. Az én ötletem az alábbi módon fest: vegyünk fel a seeker mellett egy új gombot, ami megnyit a videó képe helyén egy új nézetet, ahová kirajzolhatjuk a teljes, megjelenítési eszköztől függően egérrel, kézzel, billentyűzettel vagy mondjuk joystick segítségével navigálható gráfot, amin keresztül a felhasználó szabadon navigálhat az elágazások végtelen rengetegében, de lényeges megjegyezni, hogy a bookmarkok esetén definiált szabályok megtartása mellett, ellenkező esetben a videót elkészítő művész által megadott lejátszási folyamat integritása ugyanúgy sérülne, mint korábban, ami semmiképp sem egy elfogadható eshetőség.

Eddig már többször beszéltem a videó navigációs gráfjáról, de eddig még semmit sem mondtam arról, hogy ezt pontosan hogyan is kéne elképzelni, most ezt szeretném pótolni. Annak érdekében, hogy a több irányba szétágazó videó jól ábrázolható legyen, a gráf csúcspontjai az egyes elágazási és ugrási pontok, valamint a videó kezdete és vége lesznek. Az ezeket összekötő élek az ezek közötti videórészeket jelképezik, két pont között akkor megy él, ha a két csúcs a videó lejátszási folyamatában, bármelyik alternatív útvonal esetén, közvetlenül egymás után következik, tehát nincs közöttük másik ugrás

vagy elágazás, irányított módon, az időben korábbi elágazásból a későbbibe. Az élek súlyai a két szomszédos csúcspont által reprezentált frame-ek között eltelt időtartamok értékei. Egy ilyen gráfra egy példát az 5. ábra Egy idővonal és annak lejátszási gráfja láthatunk. Fontos megjegyezni, hogy az így kapott gráf bár súlyozott és irányított, azonban nem feltétlen DAG (Directed Acyclic Graph), mivel például időben visszafelé történő ugrások esetén kialakulhatnak benne körök.



5. ábra Egy idővonal és annak lejátszási gráfja

Az imént ismertetett seeker-gráf ötlete még tovább vihető egy újabb kiegészítő funkció segítségével: amikor a seeker tekerése során egy bookmarkhoz érkezünk, jelenítsük meg Z-irányban az idővonal felett a gráf releváns elágazását és engedjük is ezen irányok valamelyikébe továbbmenni a seekert, ezzel még kényelmesebbé téve az interaktív videók navigálását.

5.6.3 Lejátszási idő

Az egyértelműség kedvéért, mindenekelőtt szeretném tisztázni, hogy lejátszási idő alatt azt az időtartamot értem, amit egy videó egyetlen megtekintés során igénybe vesz.

Annak következtében, hogy a videó lejátszási folyamata nem írható le egy szimpla, egyenes vonallal, a videó lejátszási idejét sem tudjuk olyan egyszerűen megadni,

mint korábban. Sőt, ha belegondolunk, hogy a felhasználó lejátszás közben a saját ízlése szerint hozhat döntéseket és az esélyeink arra vonatkozóan, hogy előre megjósoljuk ezeket a választásokat eléggé limitáltak, a videó teljes, tényleges időtartamának meghatározása elsőre lehetetlen feladatnak tűnhet. Kiírhatnánk a teljes videósáv hosszát, de az a különféle ugrások és elágazások mellett irreálisan hosszú is lehet, hiszen az összes lehetőséget nagy valószínűséggel senki sem fogja végignézni, ezért ezt az ötletet gyakorlatilag el is vethetjük.

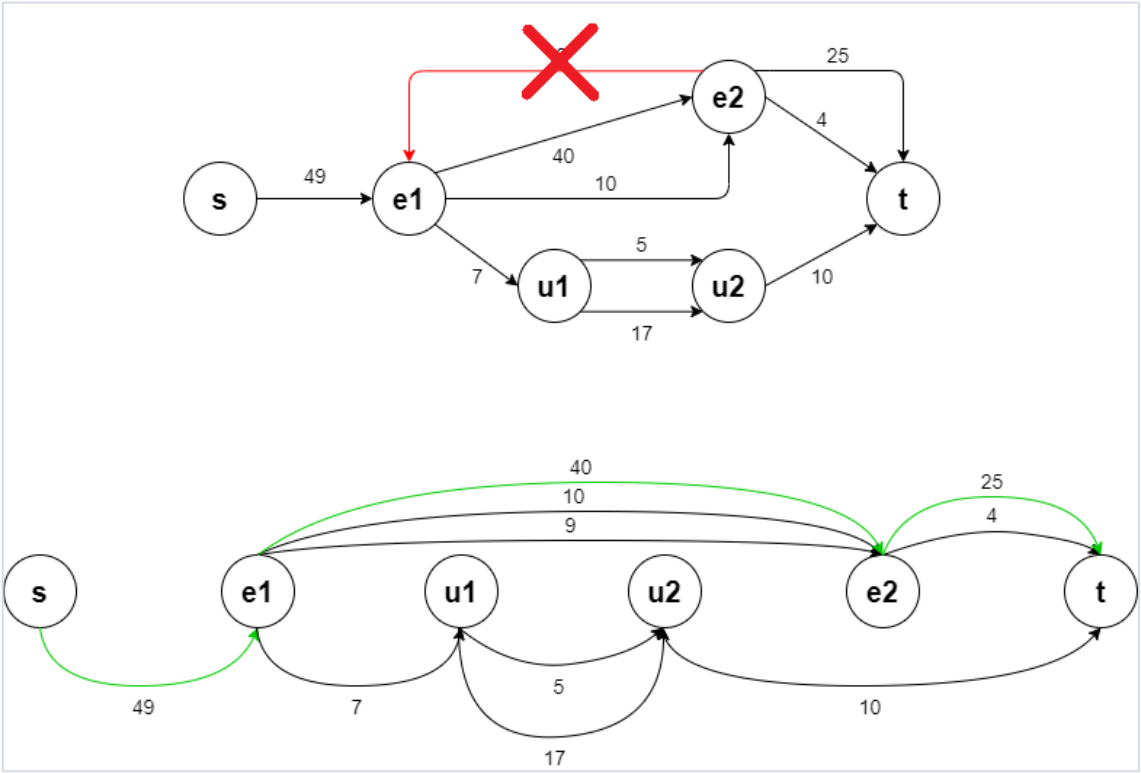
Az én megoldásom azon a gondolaton alapszik, hogy gondoljuk végig, hogy a felhasználók számára egyáltalán miért is lényeges a teljes lejátszási idő, mint információ. Ezen gondolatmenet mentén eljuthatunk arra a következtetésre, hogy arra gyakorlatilag azért van szükségünk, hogy tudjuk, hogy nagyjából mekkora időintervallumot szükséges allokálnunk egy-egy film vagy videó megtekintésére, amennyiben azt végig szeretnénk nézni. Erre alapozva az ötletem a következő: tekintsük az előző fejezetben felvázolt gráf egy „körmentesített” változatát, majd mondjuk azt, hogy a videó lejátszási ideje legyen a videó kezdetét jelölő csomópontból a videó végét jelölő gráfpontba vezető maximális hosszúságú út hossza, ez lényegében azt jelenti, hogy mondjuk azt, hogy a videó hosszának a lehető leghosszabb lehetséges lejátszási útvonal összidejét tekintjük. Az imént említett gráf előállításának és a maximális lejátszási idő megállapításának részletesebb folyamatát a következő fejezetben ismertetem részletesebben.

5.6.3.1 Maximális lejátszási idő

Annak érdekében, hogy a leghosszabb lejátszási időt megkaphassuk, először is szükséges a már megismert lejátszási gráf előállítása, ez az 5.6.2 fejezetben leírtak szerint tehető meg.

Miután megkaptuk a gráfot, azt valahogyan körmentesíteni kellene, ellenkező esetben nem leszünk képesek a leghosszabb út egyértelmű megadására, hiszen, ha van valahol kör, akkor azon tetszőleges számú alkalommal végighaladva hosszabb utat kaphatunk, mint előtte. Ennek orvoslására a következőt lehet tenni: töröljük az összes olyan élet a gráfból, amely annak egy tetszőleges csúcsából indul és egy olyan pontba mutat, amelyet a kezdőpontból indulva, tetszőleges útvonalon a kiválasztott csomópontfelé haladva már érintettünk korábban. Ezt az egyszerűbb érthetőség végett átfogalmazhatjuk a következőképpen: lejátszási idő számításakor minden olyan elágazási irányt és ugrást figyelmen kívül hagyunk, ami a videó olyan részére ugrana, ami azon

ugrási/elágazási pontig eljutva már lejátszásra kerülhetett. Gyakorlatilag ez az egyetlen olyan eset, amikor a videóban ténylegesen ismétlődéssel is járó időbeli visszaugrás következhetne be[13], vagyis kimondhatjuk, hogy az így kapott gráf már körmentes lesz.



6. ábra Az 5. ábra Egy idővonal és annak lejátszási gráfja lévő gráf körmentes változata és annak topologikus rendezése
(zöld = leghosszabb út s-ből t-be)

Ezzel a problémát sikeresen visszavezettük egy már ismert algoritmusra, mert innentől a feladat mindössze csak egy leghosszabb út keresés DAG-ban.

[13] Ezen kívül az egyetlen eset, amikor időben visszafelé történő ugrás elágazás előfordulhat az, amikor olyan videórészletre ugrunk, ami még nem kerül(hetet)t lejátszásra, ebben az esetben viszont nem alakulhat ki kör, hiszen a gráfban is egy olyan csomópontba fog mutatni az ehhez az ugráshoz tartozó él, amit egyetlen a kezdőpontból a kiválasztott pontba menő út sem érhet el, így kör sem alakulhat ki ezen él(ek) által.

Ezen algoritmus lépései nagyvonalakban a következők:

1. A gráf topologikus sorrendjének előállítása DFS (Depth-First Search) eljárás segítségével. Ennek egy lehetséges kimenetét a 6. ábra Az 5. ábra Egy idővonal és annak lejátszási gráfja lévő gráf körmentes változata és annak topologikus rendezése (zöld = leghosszabb út s-ből t-be) illusztrálja.
2. Topologikus sorrend bejárása annak elejétől a végéig.
3. Vegyünk egy kiindulási pontot a kereséshez ahonnét a távolságot számolni szeretnénk, jelen esetben ez legyen a videó elejét jelölő node, nevezzük ezt el *s*-nek. Ha az adott csomópont a sorrendben ezen *s* pont előtt van, akkor az attól vett leghosszabb távolsága legyen negatív végtelen (gyakorlatban ez valamilyen nagyon kicsi értéket jelent, pl. integer minimuma), magának a pontnak a távolsága 0 és ezekben az esetekben menjünk a következő elemre, egyébként folytassuk az eljárást.
4. Menjünk végig a gráf összes, a topologikus rendezésben ezt megelőző csúcsán és nézzük meg, hogy vezet-e onnan él ebbe a pontba. Amennyiben igen, akkor ellenőrizzük, hogy azon node *s*-től vett leghosszabb távolsága + az aktuális él hossza[14] nagyobb-e, mint a beérkezési pont aktuális távolság értéke, ha igen, akkor felülírjuk az újjal, egyébként marad a korábbi összeg.
5. Ismételjük ezt addig, amíg a csomópontot, ami és *s* között távolságot szeretnénk számolni (hívjuk ezt *t*-nek), el nem érjük. Ekkor az *s*-ből *t*-be vezető leghosszabb út hossza, azaz most éppen a maximális lejátszási idő, a *t*-beli távolságérték lesz.

5.6.3.2 Aktuális lejátszási idő

Eddig amikor lejátszására időről beszéltem, akkor mindig annak a várható, teljes videóra vonatkoztatott értékére gondoltam, pedig a legtöbb lejátszó szoftverben egy videó megjelenítése során nemcsak a teljes időtartamot, hanem azt az aktuális időt is láthatjuk,

[14] Ez most csak az él csúcsától a célcsomópontig vezető út hossza, az egyes ugrások időköltsége 0-nak tekinthető.

ahol a lejátszási folyamat éppen tart. Ezt egyrészt megtehetnénk a videó lejátszási irányainak meghatározásához létrehozott gráf[15] folyamatos végigkövetésével, így pontosan tudhatnánk, hogy hol járunk időben és a lejátszás „térképén", másrészt egy időzítő segítségével másodpercenként periodikusan frissíthetnénk az aktuális lejátszási idő értékét, amíg a lejátszás aktív. Nem nehéz látni, hogy az utóbbi megközelítés mind komplexitás, mind erőforráshasználat szempontjából előnyösebb, ráadásul webes környezetben lényegében minden adott, hogy az szinte triviális módon implementálható legyen, bár asztali alkalmazások esetén sem kéne sokkal több munka hozzá. Mindezek után azt hiszem nem meglepő, hogy a saját lejátszómban az utóbbi megoldást alkalmaztam.

Ezen koncepció esetén két megfontolandó dolog merülhet fel. Egyrészt, amikor a lejátszás megáll, akkor a lejátszási időt sem szabad megnövelni, amíg az újra nem indul, ellenkező esetben a számolt érték teljesen inkonzisztenssé válik a tényleges időhöz képes, ezáltal teljesen használhatatlanná téve azt. Másrészt olyan időpontra történő visszaugráskor, ahol már jártunk, figyelni kell arra, hogy a két időpont közötti időkülönbséget le kell vonni az aktuális lejátszási időből, ennek megállapításához viszont mindenképp szükséges használnunk a videó lejátszási gráfját, ez a két csomópont közötti élek súlyainak egyszerű végigolvasásával és azok szummázásával könnyedén megtehető, mivel a már megtekintett elágazásokat és az azokban választott továbbhaladási irányokat úgyis tároljuk, ahogy ez majd a következő fejezetből látható lesz.

5.7 Megtekintési folyamat követése

Ez a rész lényegében egy, még az előző, 5.6 fejezethez kapcsolódó kiegészítő, kényelmi funkcionalitást tárgyal, amelynek feladata, hogy kövesse a felhasználók lejátszás során hozott döntéseit és tárolja azokat a megtekintési folyamat aktuális állapotával együtt, abból a célból, hogy azok majd később visszatölthetők legyenek, például a videó bezárása, majd újbóli megnyitása esetén, ugyanott folytatva a lejátszást, ahol az korábban abbamaradt.

A döntések tárolásához egy egyszerű tömb megfelel, amelynek perzisztenciája webes környezetben localStorage használatával (ehhez a tömb tartalmát először át kell

[15] Ez nem ugyanaz, mint a maximális lejátszási idő megállapításához használt gráf!

alakítani szöveges, például JSON formátumra), míg egy asztali lejátszó esetében annak egy fájlba történő kiírásával érhető el. Egy-egy döntés követéséhez és reprodukálásához két dolog szükséges: az elágazás azonosítója, ez lehet például a videó aktuális frame-jének száma, az adott elágazás egyik elemének azonosítója[16], vagy akár egy teljesen új, kifejezetten erre a célra létrehozott numerikus érték is, illetve az ugrás célja. Ezen két adat tárolására speciális struktúrák, objektumok hozhatók létre, amiket utána bele lehet helyezni a fenti tömbbe, azt követően, hogy a néző meghozta a döntését a kérdéses elágazásban. Fontos megjegyezni, hogy a tömb tartalmának folyamatos frissítése szükséges, mivel, ha a felhasználó visszaugrik egy már megtekintett időpontra, vagy visszatekeri a videót valamelyik elágazás elé, akkor az ezt követő összes választása semmissé válik, az azokhoz tartozó elemeket a tömbből is el kell dobni.

[16] Itt az elágazás többi komponense is könnyedén megtalálható, hiszen a típusuknak és a megjelenítési idősávjuknak meg kell egyezniük.

6 Implementáció

Ebben a fejezetben áttekintem az általam implementált szoftvereket és részletesebben ismertetem annak részleteit, elsősorban azokra a gondolatokra fókuszálva, amikről még nem esett szó.

Mielőtt bármibe is belekezdenék, szeretném konkrétan is kimondani, hogy pontosan mi az, amit megvalósítottam, mert bár ezekre korábban már többször is utaltam, azok konkrét listázása még nem történt meg, tehát az implementált alkalmazások:

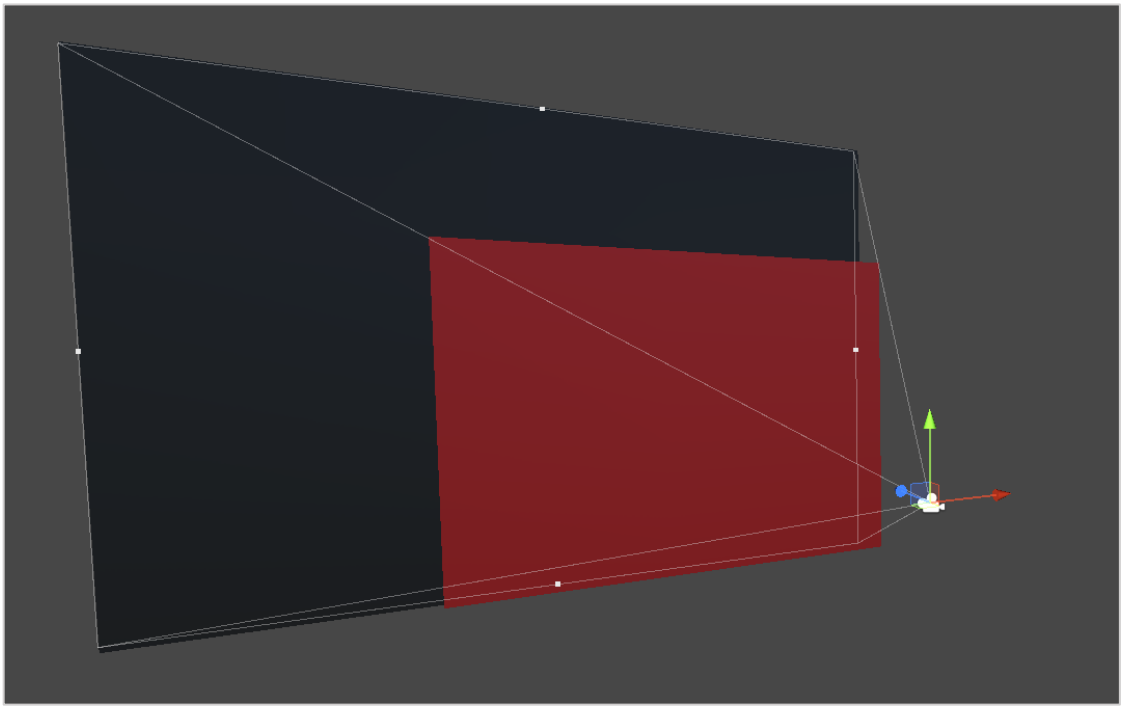
- **Webes lejátszó:** egy HTML5 alapú, JavaScript Video API-ra építő videólejátszó, ami képes interaktív videók létrehozására, az interakciós adatokat külső fájlból nyerve.
- **Unity-alapú lejátszó:** ténylegesen nem készítettem el, de a környezetet és annak implementációs lehetőségeit megvizsgáltam.
- **H.265 videó enkóder és dekóder:** itt megvizsgáltam a videószerkesztő- és lejátszó szoftverek alacsonyabb rétegeit is. A félreértések elkerülése végett, nem teljes értékű enkóder/dekóder szoftvert készítettem el, hanem már létező szoftvereket egészítettem ki az interaktív videólejátszást támogató funkcionalitással.
- **Natron videószerkesztő plugin:** az említett videószerkesztő (pontosabban kompozitáló) eszközhöz készítettem egy[17] plugint, ami előállítja a megfelelő kimenetet az előtte elkészített webes lejátszómhoz.

6.1 Unity-alapú konstrukció

A Unity nevű játékmotort azt hiszem ma már senkinek nem kell bemutatni, ezt választottam annak vizsgálatára, hogy az általam elképzelt koncepció egy játékmotorban is megvalósítható lenne-e. A válasz az, hogy az elképzelés nagyon is működőképes, maga a megvalósítás a következő lépésekből áll:

[17] Helyesebben kettőt, további részletek a 6.4.3 fejezetben olvashatók.

1. Új színtér felvétele, abban elég a kamerát megtartani, ezen felül szükség lesz egy új GameObjectre egy hozzáadott VideoPlayer komponenssel.
2. A VideoPlayer rendereljen a kamera hátsó vágósíkjára és a képaránya legyen nyújtott. Ezzel többé-kevésbé ugyanúgy fog viselkedni a lebuildelt alkalmazás, mint egy átlagos asztali videólejátszó.
3. A videólejátszón legyen egy script, aminek a feladata az interaktív elemek megfelelő időpontokban történő megjelenítése, a kattintások észlelése és azok alapján az interakciók kezelése.
4. A VideoPlayer, amennyiben igény van rá, kiegészíthető még egy scripttel, ami a szokásos videó irányítási funkciókat (pl. szünet, hangerő, seeker) implementálja.



7. ábra A fenti koncepció szimbolikus vizualizációja (fekete = videó, piros = interakciós réteg)

Ez már elég egy interaktív videólejátszónak a Unity játék motorban történő elkészítéséhez. Annak, hogy ezzel itt, koncepció szintjén meg is álltam az az oka, hogy a VideoPlayer dokumentációját [11] részletesen áttekintve azt találtam, hogy az gyakorlatilag mindent támogat, amit a JavaScript Video API is, így lényegében ugyanazt a kódot kellett volna itt is megírni, mint a webes lejátszó esetén, csak egy másik nyelven, más elnevezésekkel, bármiféle koncepcionális különbségek nélkül. Ezutóbbi lejátszó implementációját mélyrehatóbban a következő fejezetben tárgyalom.

6.2 Webes konstrukció

A web a HTML5 megjelenésével kényelmes megoldásokat nyújt videók megjelenítésére, ezt mi sem bizonyítja jobban, hogy az eddig elkészült interaktív technológiák is elsősorban ezt a platformot célozták meg. Munkám során egy lejátszó elkészítésével magam is megvizsgáltam a webes környezetben elérhető lehetőségeket, egyrészt videók lejátszására általában, másrészt ezekhez interaktivitás hozzáadására.

6.2.1 JavaScript Video API

A HTML5-ös videóelemeket tipikusan JavaScriptből tudjuk vezérelni, erre a Video, vagy általánosabban MediaElement[18] elnevezésű API ad lehetőséget. Az API lehetőséget biztosít nekünk a videó alapvető tulajdonságainak (pl. forrás URL, szélesség, magasság, vezérlők megjelenítése, …) olvasására és módosítására. Ezen felül módunk nyílik az API event hookjainak használatára (pl. videó betöltődött, lejátszás elindult, …), és a videó állapotának vezérlésére (pl. szünet, aktuális idő beállítása, újratöltés, …).

Végeredményben a HTML5 már egy kész megoldást ad videók lejátszására, a beépített opciókkal is le tudunk egy videót egy weboldalon játszani, viszont a fenti API ereje abban rejlik, hogy lehetőséget teremt az alapvető működések felülbírálására, és lényegében akár egy saját videólejátszó szoftver írására azon keresztül.

Munkám során a MediaElement API utóbbi tulajdonságára építve, először létrehoztam egy új lejátszó felületet, elsősorban abból a célból, hogy mélyebben is megismerkedhessek magával az API-val, illetve amennyiben az interaktivitás implementálása során szükség lett volna az alapvető funkciók módosítására, kiegészítésére, akkor azt sokkal egyszerűbb lett volna a saját kódomban megtenni. Szerencsére, a megvalósítás során ez az igény nem merült fel, az interaktív funkcionalitást sikerült egy teljesen független, külön modulként elkészíteni, a következő fejezetben pedig azt is részletezem, hogy pontosan hogyan.

6.2.2 Implementáció

Ebben a részben bemutatom az általam elkészített, webes környezetben használható videólejátszó megvalósításának főbb lépéseit, kezdetben áttekintve magát a

[18] Azért általánosabb, mert ebben már az Audio API-val közös részek is benne vannak. [12]

projektet és annak struktúráját, majd innen lépésenként haladva a lejátszó részletein keresztül, eljutva az interaktivitás implementációs részleteiig.

6.2.2.1 Projekt felépítése

Maga a projekt alapvetően három fő komponensre osztható fel, ezek:

- **HTML-leírás:** a lejátszó struktúrájának HTML alapú leírásra.
- **Videólejátszó:** a JavaScript Video API-ra épített saját lejátszó modul, ami gyakorlatilag egy egyszerűbb, hagyományos videólejátszót implementál, interaktív funkcionalitás nélkül.
- **Interaktív modul:** egy különálló JavaScript fájl, ami interaktív funkcionalitással egészíti ki a videólejátszót.

A következő részekben ezeket fogom egyenként áttekinteni.

6.2.2.2 HTML struktúra

A videólejátszó szerkezete HTML-ben leírva az alábbi módon fest (az interakciók kezeléséhez szükséges elemekkel együtt):

```
<div id="video_wrapper">
    <video width="100%" height="100%" id="interactive_video">
        <source src="./video.mp4" type="video/mp4">
    </video>
    <div id="toolbar">
        <div id="seeker">
            <div id="seeker_progress"></div>
            <div id="currentTime" class="seeker-time-display"></div>
            <div id="fullTime" class="seeker-time-display"></div>
        </div>
        <div id="controls_row_one">
            <div id="main_controls">
                <input type="button" value="Play" id="play" />
                <input type="button" value="Pause" id="pause" />
                <input type="button" value="Stop" id="stop" />
            </div>
            <div id="side_controls">
                <input type="range" min="0" max="1" step="0.01" value="0.5"
                 id="volume" />
                <input type="button" value="Full Screen" id="fscreen" />
            </div>
        </div>
        <div id="interactive_controls">
            Interactive data: <input type="file" id="interactive_input"
                              accept=".txt" />
        </div>
    </div>
    <div id="video_background_render_wrapper"></div>
</div>
```

A kódból a következőket olvashatjuk ki:

- Három fő alkotóelem van: maga a videó, a toolbar, ami tartalmazza az összes, a videó irányítására használható elemet és a „background render wrapper”, ami lejátszás közben magában foglalja az összes pufferelés céljából létrehozott, háttérbeli videóelemet a 5.5.2 fejezetben ismertetett elv alapján, így azok egyszerűen nyilvántarthatók és kezelhetők.
- A seeker is három részre osztható: maga az idővonal, az aktuális lejátszási idő és a teljes lejátszási idő, ezek értékei a 5.6.3 szakaszban ismertetettek szerint frissülnek.
- Három a videó lejátszását vezérlő elem található a felületen: lejátszás, szünet (a videó az aktuális időponton marad, de a lejátszás leáll) és teljes megállítás (a lejátszás félbeszakad és visszaugrik a videó kezdetére). Ezen felül még lehetne további funkciógombokat is beszúrni, mint például a videó előre/visszatekerése, ahogy azt sok más szoftverben is láthattuk, de a videólejátszó alapvetően inkább demonstrációs célokat szolgál, minthogy teljeskörű lejátszási funkcionalitást nyújtson és ezutóbbi egyébként is elérhető a seekerről, úgyhogy ilyen jellegű kiterjesztésekkel nem foglalkoztam.
- A felületen megtalálható standard videóelemek még a hangerőszabályzó és a teljes képernyős lejátszás ki/bekapcsolásának lehetősége.
- Akad egy új, nem szokványos komponens is: ez egy fájl beviteli mező, amivel az interaktív elemeket leíró információkat tartalmazó szöveges fájlokat lehet betölteni, tulajdonképpen a lejátszóban aktuálisan megjelenített videó ezen fájl beolvasása után válik interaktívvá (mivel webes környezetben egyelőre nincs lehetőség az adatoknak közvetlenül a videó streamből történő olvasására).

A fenti kód által leírt videólejátszó felületének megjelenését a 4. ábra Bookmarkok megjelenítése a seekeren láthatjuk lejátszás közben.

6.2.2.3 Videólejátszó

Itt mindenekelőtt szeretném megjegyezni, hogy annak okán, hogy a videólejátszás során a valósidejű, minimális válaszidők elérése kritikus jelentőséggel bír, az

implementáció során kizárólag a natív JavaScript képességeire hagyatkoztam, bármiféle külső könyvtár nélkül, a maximális teljesítmény kiaknázása, valamint a lehető legtöbb elérhető platform támogatása érdekében, hasonlóan ahhoz ahogy például az asztali környezetben vizsgált enkóder és dekóder szoftverek legnagyobb része is ANSI C/C++-ban, illetve egyes modulok esetén Assemblyben íródott.

A lejátszó kódjának nagy része eseménykezelőkből áll, amelyek az előző fejezetben bemutatott egyes elemekkel történő interakciókat kezelik a Video API-n keresztül, gyakorlatilag egyfajta burkoló réteget képezve afölött. Sajnos a lejátszási folyamat felett ennél mélyebb irányítást standard eszközökkel egyelőre nem tudunk szerezni, legalábbis amíg a WebCodes API [10] hivatalosan is elérhetővé és az egyes böngészők által támogatottá nem válik.

Az eseménykezelőkön kívül a videólejátszónak két másik része is akad: egy inicializációs rész, ahol gondoskodom a videó megfelelő betöltéséről és az egyes eseménykezelők is itt kerülnek beregisztrálásra, valamint egy Update() függvény, amely a játékok ideológiájához hasonlóan folyamatosan meghívódik a videó lejátszása közben[19]. Ezutóbbi metódus a seeker adatainak (seeker bar, lejátszási idők) naprakészen tartásáért felel.

6.2.2.4 Interaktív modul

A kód struktúrája több hasonlóságot is mutat a videólejátszójával, itt is több eseménykezelőt, valamint egy incializálást és egy periodikus frissítést találunk, azonban itt a szituáció már jelentősen árnyaltabb. Itt kezdetben a feladat a videó betöltése helyett az esetlegesen mentett megtekintési folyamat és aközben hozott döntések localStorage-ból történő kiolvasása, továbbá azok JSON-ben tárolt tartalmának parse-olása, míg a tényleges inicializációs folyamat csak az interaktivitási adatokat tartalmazó fájl megadása után indul el, ennek főbb lépései az alábbiak:

1. Az egyes interaktív elemek beolvasása egy tömbbe (Koordináták, megjelenítési időintervallum, interakció típusa, tartalom, …)

[19] Sajnálatos módon ez nem frame-enként hívódik, de még csak nem is fix időközönként, helyette az irányítás teljeskörűen a böngészők kezében van, hogy milyen frekvenciával triggerelik a videó frissítési eseményét.

2. Az egyes interaktív elemeknek megfelelő HTML node-ok létrehozása és a DOM-hoz (Document Object Model, nagyon nagy vonalakban nézve, az oldal aktuális struktúrájának leírása) történő hozzáadása úgy, hogy azok egyelőre ne legyenek megjelenítve.
3. Az elágazásokhoz bookmarkok létrehozása, valamint a seeker kattintásokat ezen bookmarkok után blokkoló elemek felvétele.

Ezen kívül ez a modul is tartalmaz egy update() funkcionalitást megvalósító metódust, aminek a feladatai a következők:

- A lejátszási gráf (lásd.: 5.6.2 fejezet) felhasználásával localStorage-ból az összes olyan döntés törlése, ami létezik, de időben még előtte járunk, annak, hogy ez az állapot bekövetkezik többféle oka lehet, például a néző visszatekerte a videót, vagy egy időben visszafelé történő ugrást hajtottunk végre, ezért ezzel érdemes itt foglalkozni és nem eseménykezelőkben.
- Az egyes interaktív elemekhez a specifikus viselkedések kezelése, mint mondjuk az automatikus ugrások, vagy elágazások esetén a videó megállítása/loopolása.
- Aktív elágazás nyilvántartása és ennek frissítése.
- Pufferelő háttérbeli videóelemek létrehozása elágazásba történő belépés esetén és törlése, amikor kijutottunk onnan.
- Az egyes interaktív elemek megjelenítése, illetve elrejtése a megjelenítési intervallumuknak megfelelően.
- Az éppen látható interaktív komponensek animációja (transzformációja).
- Az egyes bookmarkok utáni kattintás-blokkoló elemek eltüntetése, ha továbblépünk a könyvjelzőhöz tartozó elágazáson és felfedése, ha valamiért időben visszamegyünk az elágazás elé, itt ismét a lejátszási gráfot használva ennek ellenőrzésére.
- Seekeren látható idők frissítése, a videólejátszót felülírva, az 5.6.3 fejezetben leírtakat követve.

Megemlíteném még, hogy az egyes interakciós típusokat egy enum-mal kezelem, ami JavaScript esetében lényegében egy konstans objektumot jelent, amelynek kizárólag

olyan változói léteznek, amilyen értékei annak előfordulhatnának, ha ténylegesen egy enum lenne, mint más nyelvek esetén, ezáltal az interakciós típusok könnyedén bővíthetők.

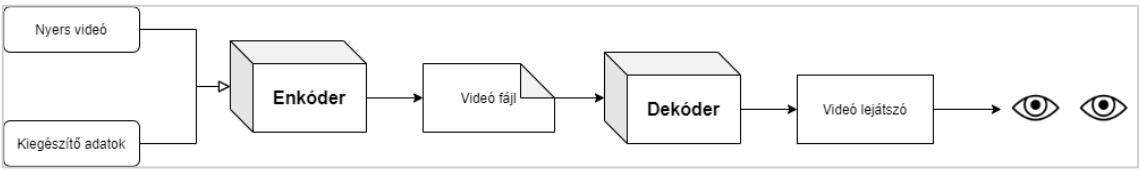
Az utolsó lényegesebb gondolat, amiről még itt szeretnék beszélni az ugrások kezelése, ezalatt az összes időbeli ugrást általánosságban értem, történjenek azok elágazásból, egyszerű ugrásként, vagy egy automatikus ugrás részeként. Ennek folyamata a következőképpen zajlik:

1. Videó pillanatnyi szüneteltetése, hogy az ugrás ne okozzon esetleges nem várt mellékhatásokat.
2. Ugrás végrehajtása az aktuális idő átállításával.
3. Ha van aktív elágazás és abból kiugrunk (tehát nem azért ugrunk, mert pont az elágazásban szeretnénk loopolni), akkor elmentjük az ott hozott választást localStorage-be, egyébként ugrás a 6. lépésre.
4. Az összes ehhez az elágazáshoz tartozó elágazás elem bookmark blokkolójának eltüntetése.
5. Aktív elágazás deaktiválása.
6. Lejátszás folytatása, ha az nincs megtiltva, ami például olyan esetben fordulhat elő, ha a felhasználó időben egy még meg nem tekintett elágazás után szeretne navigálni és ahelyett visszaugrunk az utolsó elágazásra.

Ezzel lényegében az interaktív funkcionalitás összes fontosabb kérdését sikerült kezelni a videólejátszó oldaláról, az készen áll a használatra.

6.3 H.265 alapú konstrukció

Ebben a szakaszban nagy vonalakban áttekintem, hogy hogyan is néz ki egy az adatokat videó stream-ben SEI üzenetek segítségével tároló, interaktív videók lejátszására képes konstrukció.



8. ábra Videó kódolási és lejátszási folyamat áttekintése

Mint az a 8. ábra Videó kódolási és lejátszási folyamat áttekintéseán is látszik, a folyamat a következőképpen néz ki:

1. **Enkóder bemenetének létrehozása:** ez egy nyers (bináris) videó és a hozzátartozó kiegészítő adatok (pl. SEI üzenetek) összessége. Ezek létrejöhetnek például egy videószerkesztő szoftver kimeneteként.

2. **Kódolás az enkóderrel:** az enkóder a bemeneti adatok alapján valamilyen kodek (pl. H.265) segítségével létrehoz egy jól ismert videó fájlt (pl. .mp4).

3. **Dekódolás dekóderrel:** az enkóder által létrehozott videó fájl megtekintéskor egy dekóder bemenete lesz.

4. **Videó lejátszása:** a dekóder kimenete már közvetlenül is megjeleníthető egy videólejátszóban.

A fenti folyamatnak két kulcsfontosságú eleme van, amit programozni is tudunk, ezek: az enkóder és a dekóder[20]. Ezek nulláról történő megírása túlmutat a diplomaterv keretein, ezért a saját megoldásomat nyílt forrású szoftvereket felhasználva készítettem el.

6.3.1 Enkóder

Ez a szoftver felel azért, hogy a nyers bemeneti adatokból egy lejátszható videó fájl jöjjön létre. A saját megoldásomat a nyílt forrású x265 projektre [13] építettem, amely valószínűleg a legnépszerűbb H.265 enkóder.

Az enkóderben az interaktív területek szempontjából a cél kettős: egyrészt az, hogy az egyes kattintható felületek adatai bekerüljenek a videóba, azon belül is a megfelelő frame-ekbe, másrészt az, hogy a kódolás során a képkockák megfelelő sorrendben kerüljenek bele a videóba ahhoz, hogy az utána hatékonyan dekódolható legyen interakciók mellett is. Ennek a folyamata a következőképp történik:

1. Az interaktív területekre vonatkozó bemeneti adatok egy külön fájlban kerülnek létrehozásra, amit meg kell adni az enkóder bemeneteként, ami majd azt beolvassa és feldolgozza. A bemeneti adatok gyakorlatilag megegyeznek a kódoltakkal,

[20] Illetve van még egy harmadik is, a lejátszó, de ezzel már a webes fejezetben foglalkoztam, az elvek itt is ugyanazok, csak az információt máshonnét kell kiolvasni.

amiket a következő fejezetben ismertetek. Azokon felül az egyedüli extra adatot a POC számok (lásd. alább) képviselik.

2. Az egyes képkockák feldolgozásakor megvizsgáljuk, hogy van-e az adott kép POC-jával megegyező POC-kkal rendelkező interaktív adat, ha van, akkor azt beírjuk az adott NAL AU-hoz tartozó SEI üzenetek közé a 6.3.1.1 fejezetben ismertetett formátumban.

POC (Picture Order Count) = egy-egy képkocka sorszáma a videófolyamban, felfogható úgy is, mint az egyedi azonosítója.

3. A feldolgozás végeztével az interaktivitási információk bekerültek a videó stream-be.

6.3.1.1 Kódolt adatok

Jelenleg minden egyes objektumhoz az alábbi adatok kerülnek egy SEI üzenetben enkódolásra:

- Az interaktív felület egyedi azonosítója (ez a bemeneti fájlban van specifikálva minden egyes objektumhoz).
- Interakcióra típusa egy számmal (enummal) azonosítva, ennek lehetséges értékei megegyeznek a 5.2 fejezetben tárgyaltakkal.
- Az interakciója célja, pl. ugrás esetén annak a frame-nek az azonosítója, ahová ugrani szeretnénk.
- Az alakzat ebben a frame-ben először kerül-e definiálásra?
    - Amennyiben igen, akkor az alakzat típusa (pl. négyzet, kör, görbe stb.) és annak koordinátái (pl. kör esetén a középpontja és a sugara).
    - Amennyiben nem, az alakzatot nem definiálom még egyszer, helyette egy transzformációs mátrixot adok meg, amivel az adott alakzat az aktuális frame-ben az eredeti pozíciójához és orientációjához képest transzformálva van.

6.3.1.2 Pufferelés

Normál videólejátszás esetén egyszerű dolgunk van, az egyes képkockákat elég csak sorban belehelyezni a videó stream-be, azonban ez interaktív videók esetén nem

feltétlen járható út, hiszen az egyes elágazásoknál nem tudhatjuk előre merre megy majd tovább a videó, ami annyit jelent, hogy ha rosszul döntünk, azaz nem megfelelő sorrendben pufferelјük a frame-eket, akkor könnyen kerülhetünk olyan helyzetbe, hogy a videó mégsem arra megy amerre gondoltuk. Ennek eredményeként a pufferelt képkockákat kidobhatjuk a kukába és a lejátszás is esélyes, hogy egy rövid időre meg fog akadni, amíg nem érjük újra utol a videót a pufferben.

Az alábbiakban ezen probléma kiküszöbölésére ismertetnék egy lehetséges ötletet.

Induljunk ki abból, hogy minden egyes elágazásnál ismerjük az összes lehetséges irányt, amerre a lejátszás továbbmehet. Minden egyes elágazó frame-nél helyezzük el az összes olyan képkockát a DPB-be ahová lehetséges, hogy ugrani fogunk (illetve amíg a DPB kapacitása engedi az ezeket követőket is). Ugrásnál ugorjunk a megadott frame-re és dobjuk el az összes másik irányhoz tartozó képkockát a DPB-ből. Ezt követően már csak az adott irányhoz tartozó képkockákat fogjuk pufferelni a DPB-be.

Ezzel végeredményben elértem azt, hogy egy-egy elágazásnál bármilyen irányba is megy tovább a videólejátszás, mindig lesznek elérhető, előre pufferelt frame-ek.

DPB (Decoded Picture Buffer) = a dekódolt képeket tartalmazó puffer, tartalmát az enkóder határozza meg, ami többek között azt is megmondhatja képről-képre, hogy milyen frame-eket szeretnénk megtartani, avagy eldobni belőle. A kapacitása felbontástól függően 6, 8, 12 vagy 16 kép (az aktuálisan megjelenített képet is beleszámítva).

Látható, hogy bár a fenti megoldás elméletileg orvosolja az eredeti problémát, bizonyos limitációkat is bevezet abban a tekintetben, hogy egy-egy frame-ről maximális 8192x4320-as felbontás esetén maximum 5, míg legjobb esetben is legfeljebb 15 ugrás lehetséges, hiszen az egyes alternatív irányoknak is bele kell férniük a DPB-be.

6.3.2 Dekóder

Ennek a szoftvernek a feladata, hogy egy videó fájl tartalmát egy videólejátszó által lejátszható formára alakítsa, gyakorlatilag az enkóder inverze és olyan szinten hasonlít rá, hogy azonos adatstruktúrákat használnak és lényegében maga a kód is megegyezik, csak itt írás helyett olvasás történik. A saját megoldásomban a libde265 [14] nyílt forrású könyvtárat használtam.

A dekóder feladata interaktív területek szempontjából viszonylag egyszerű: az enkóder által kódolt adatok dekódolása, a tényleges interakciókat már a videólejátszó kell, hogy kezelje, amire egy példát – igaz, nem a videó streambe kódolt adatokkal – a webes implementáció esetén (6.2 fejezet) láthattunk.

Összességében ezzel egy videó lejátszásához szükséges összes főbb komponenst megvizsgáltam az interaktív funkcionalitás szempontjából, már csak a létrehozás folyamatából hiányzik egy alkotóelem, amit a következő fejezetben ismertetek.

6.4 Szerkesztő szoftver

Ezen a ponton, elkészült egy webes lejátszó szoftver, illetve egy H.265 kodeket támogató enkóder/dekóder, ami interaktív videók kódolásához is használható, viszont továbbra is hiányzik egy olyan megoldás, amellyel egy átlagos felhasználó, vagy művész képes lehetne interaktív videók létrehozására. Ennek orvoslására, egy ismertebb szerkesztő programhoz lehetne hozzáadni a szükséges funkcionalitást. Az alábbiakban keresek egy erre megfelelő szoftvert, ismertetem annak sajátosságait, majd bemutatom hogyan lehet abban implementálni interaktív videók készítésének támogatását.

6.4.1 Elvárások

Alapvetően négy fő elvárásom lenne a fejlesztési alapként használt szerkesztővel kapcsolatban:

- Annak érdekében, hogy a felhasználók számára könnyen megszokható, megismerhető legyen, érdemes lenne valamelyik népszerűbb alkalmazásra építeni.
- A program rendelkezzen a fejlesztői részéről megfelelő támogatással, legyen még ma is aktív fejlesztés alatt.
- A szoftver legyen nyílt forrású, vagy rendelkezzen plugin készítési lehetőségekkel.
- Létezzen hozzá egy részletes, jól karbantartott dokumentáció, ne annak hiánya miatt legyen feleslegesen nehézkes, időigényes a fejlesztés, mint az tapasztalataim alapján egyébként sok nyílt forrású szoftverre jellemző.

Ezen szempontok alapján több videószerkesztőt is megvizsgáltam, az eredményeket az itt látható táblázatokban gyűjtöttem össze.

	OpenShot	**Shotcut**	**Adobe After Effects**
Előnyök	Népszerű, nyílt forrású, jól dokumentált C++ API plugin készítéshez	Viszonylag népszerű, nyílt forrású	Népszerű, jól használható C++ API plugin készítéshez, részletes dokumentáció
Hátrányok	Az API elsősorban a tényleges videó- és audiósáv manipulációjára lett tervezve, más jellegű extra funkcionalitás hozzáadása problémás lehet	Gyenge plugin támogatás, erősen hiányos dokumentáció	Nem nyílt forrású, fizetős, drága, emiatt nehezen hozzáférhető

	Vidcutter	**Blender**	**LiVES**	**Natron**
Előnyök	Egyszerű, nyílt forrású	Népszerű, nyílt forrású, videószerkesztést is támogat, viszonylag jól dokumentált Python API addon készítéshez	Relatíve ismertebbnek számít, nyílt forrású	Népszerű, nyílt forrású, jól dokumentált Python és OpenFX (C++) API plugin készítéshez
Hátrányok	Nincs plugin támogatás, az esetleges módosításokhoz mindig újra kell fordítani a teljes kódbázist	Láthatóan nem videószerkesztésre lett létrehozva és ez az API-n is meglátszik, emiatt nem igazán a megfelelő eszköz a célra	Egyre lassuló új fejlesztések, gyenge plugin támogatás, hosszú ideje nem frissített dokumentáció	Elsődlegesen kompozitáló szoftver, nem szerkesztő, szokatlanabb, node-alapú rendszerrel

A fenti szempontok és az egyes szoftverek kipróbálása után végül a Natront választottam.

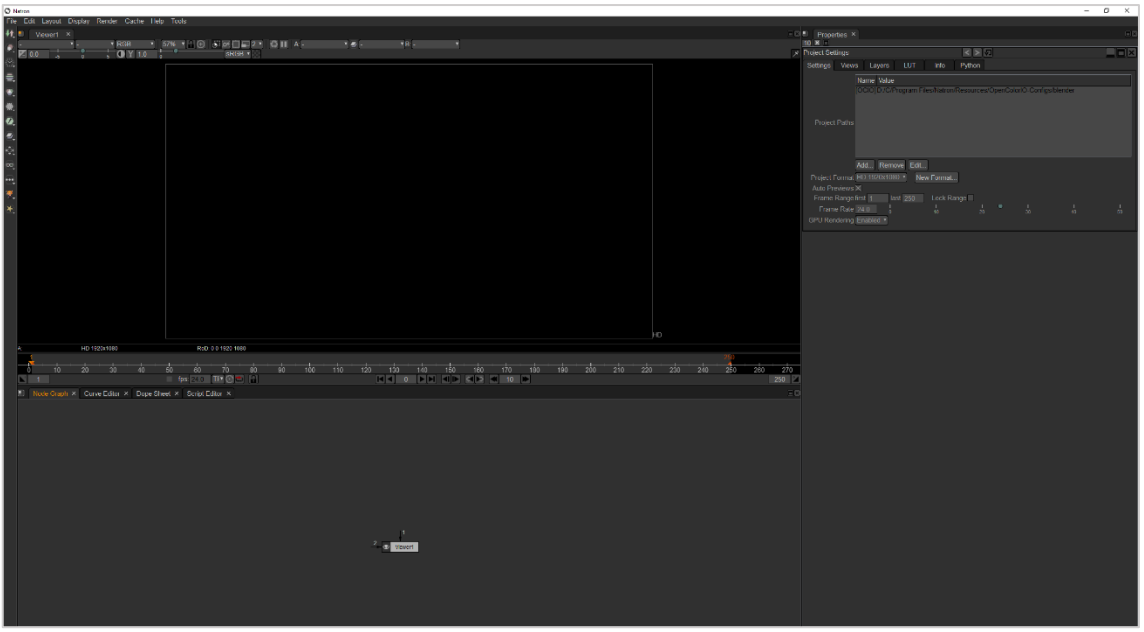
6.4.2 Natron

A Natron egy nyílt forrású videó kompozitáló szoftver, amely az összes nagyobb asztali platformon (Windows, Linux, macOS, FreeBSD) futtatható. Sajátossága, hogy 2D és 2.5D videószerkesztést is támogat és egy, a videószerkesztő szoftverekre kevésbé jellemző node-alapú rendszerrel dolgozik. Ebben az alfejezetben ezt az alkalmazást szeretném részletesebben is bemutatni.

6.4.2.1 Általános ismertető

A Natron, mint azt már említettem egy videó kompozitáló alkalmazás, ez lényegében annyit jelent, hogy a videószerkesztőkkel ellentétben, ahol a hangsúly legtöbbször a vágáson van, itt az meg sem jelenik funkcióként (legalábbis beépítetten), hanem helyette több képet/videót, vagy más audiovizuális elemeket tudunk összegyúrni eggyé, majd azokon különféle utómunkákat végezni.

Ha megnyitjuk az alkalmazást, akkor az alábbihoz hasonló kép fogad minket:



9. ábra A Natron felülete

Mint látható, a felület panelekből épül fel. Felül található a menü, azalatt pedig egy előnézet ablak, ahol a készülő videót aktuális állapotában meg tudjuk tekinteni, ezen panel tetején annak beállításait, alján annak idővonalát láthatjuk, aminek a segítségével a videóban navigálhatunk, alapesetben frame-ek szerinti felbontásban. A szoftver ablakának bal szélén egy eszköztár található, aminek az egyes opciói lényegében egy-egy különböző típusú node-ot hoznak létre, magukat a node-okat és azok gráfját alul

találhatjuk meg, ezekről részletesebben a következő fejezetben lesz szó. Jobb oldalt jelenleg a projekt beállításai láthatók, valamint, ha megnyitnánk egy csomópontot, akkor annak a tulajdonságai is itt jelennének meg. Mindezeken felül, amit még érdemes megemlíteni az a Script Editor (bal alsó panelen látszik a hozzátartozó fül), ezen keresztül könnyedén tudunk debug üzeneteket megjeleníteni a saját kódunkból, illetve az esetleges hibák részletei is itt jelennek majd meg.

6.4.2.2 Node-alapú struktúra

A Natron egy, a videószerkesztő szoftverekre kevésbé jellemző node-alapú rendszerrel dolgozik, ez alapvetően úgy működik, hogy vannak *Viewer* csomópontok, amik a nézőt reprezentálják, mint kimenet. Ilyenekből akár több is lehet, például osztott képernyős videó esetén. Ezen felül léteznek különféle bemeneti node-ok, mint például a *Read*, ami egy külső fájlt (képet vagy videót) olvas be. Ezek a csomópontok egymással összeköthetők, tipikusan minden node-nak van legalább egy bemenete és egy kimenete, ezzel egyfajta fát, vagy több Viewer esetén fákat alkotva, végül pedig a Viewerek bemenetei kerülnek majd megjelenítésre. Amennyiben esetleg több effektet is szeretnénk egyszerre ugyanazon bemenetre alkalmazni, vagy több bemenetet szeretnénk valamiért eggyé olvasztani, arra a *Merge* node-okat használhatjuk. Azért, hogy az egész koncepció értelmet nyerjen, képzeljünk el egy olyan esetet, ahol van egy bemeneti Read node, amely egy videót olvas, ennek a kimenete rá van kötve egy filter node-ra, ami mondjuk kicsit elmossa a videó képét (*Blur* node), majd ennek a kimenete bemegy egy Viewerbe, ezzel elértük azt, hogy az eredeti videót elmostuk. A fenti egy egyszerű konstrukció, nyilván a szoftver ennél sokkal többre is képes, de azt hiszem ez egy megfelelő példa arra, hogy a node-ok működését és lehetőségeit megértsék azok is, akik egyébként még soha nem találkoztak hasonlóval.

6.4.2.3 Plug-in fejlesztés

A kódbázis alapvetően két különböző típusú plugin fejlesztésre is fel van készítve, egyrészt lehetőségünk van azok Pythonban történő megírására úgynevezett PyPlugok formájában, másrészt támogatja az OpenFX szabványt követő C++ alapú bővítmények fejlesztését is. A saját implementációmat Pythonban készítettem el, ennek elsősorban személyes okai vannak, ugyanis a nyelvvel eddig mindig csak érintőlegesen, eléggé limitált keretek között tudtam foglalkozni, így jó volt egy kicsit kipróbálni, hogy milyen is azt ténylegesen egy alkalmazás megírásához használni.

Python alapú pluginok írására is két lehetőségünk van: készíthetünk kisebb scripteket, ezek például kisebb menüből elérhető funkciók lehetnek, vagy akár csinálhatunk teljes értékű PyPlugokat is, amik általában egy-egy új node típust reprezentálnak, esetleg új UI ablakokat is hozzáadnak a szerkesztőhöz. Az implementációs részben majd mindkettőre láthatunk példát.

6.4.3 Implementáció

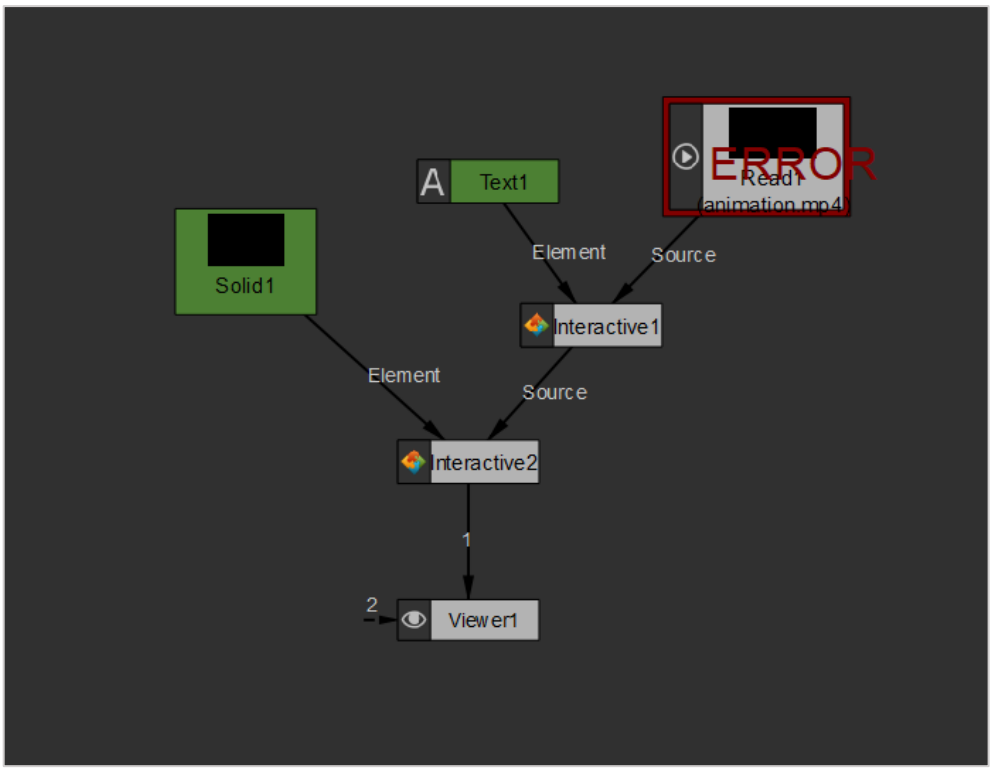
Miután egy gyors képet kaptunk magáról a Natronról, ideje az implementációról is beszélni, ebben a fejezetben annak részleteit szeretném ismertetni.

6.4.3.1 Koncepció

A szerkesztő megvalósítása során a legnagyobb kérdés az volt, hogy hogyan tudnám az interaktív funkciókat a Natron node-alapú struktúrájába beilleszteni. Erre végül a következő ötlettel álltam elő: legyen egy új node, ami tulajdonképpen csak adminisztratív funkciókat lát el: kap egy képi vagy szöveges elemet a bemenetén, ez lesz maga az interaktív elem tartalma, a csomópont tulajdonságai között beállíthatunk olyan dolgokat, mint az elem megjelenítési időtartama vagy az interakció típusa/célja és egyes speciális interakciók esetén az azokhoz szükséges paraméterek. A node kimenetén visszaadja a tartalmát, hogy az majd megjeleníthető legyen magán a videón is. Ezek után az adatok exportálása streambe történő kódolás esetén megoldható a videó renderelésekor, míg, ha külön fájlra is szükségünk van, például webes lejátszóhoz, akkor arra létrehoztam egy új menüpontot, amin keresztül ez exportálható.

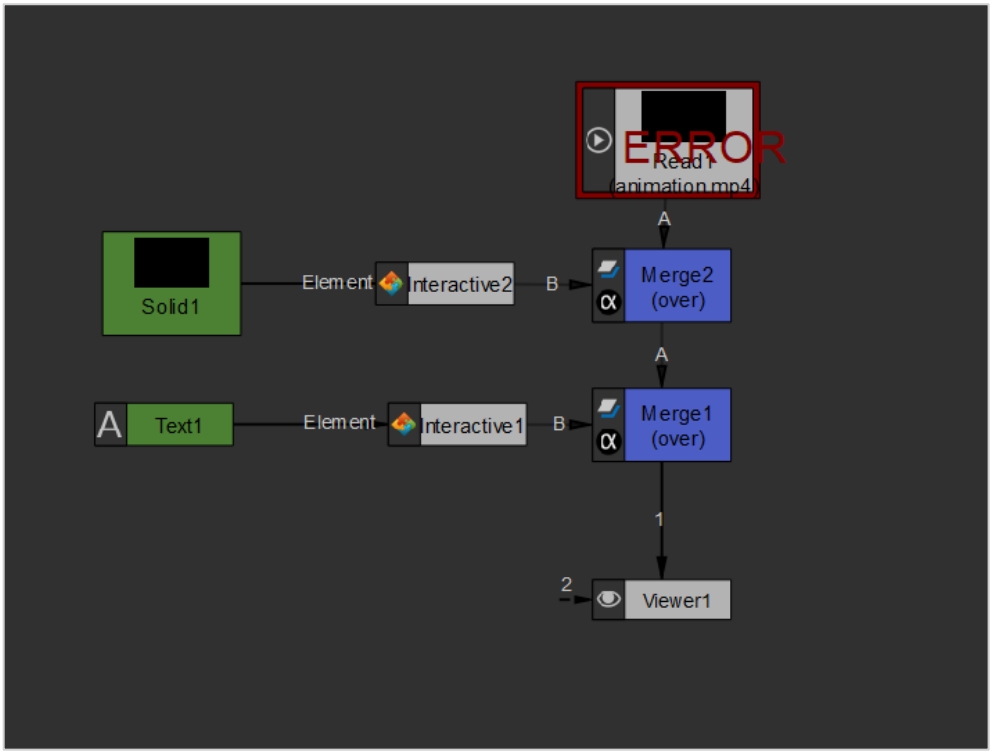
Az alapkoncepció így már megvan, viszont egy kérdés még mindig nyitott maradt: hogyan illeszkedjen az új csomópont a node-gráfba? Ezzel kapcsolatban két eltérő esetet vizsgáltam:

- Tehetjük az interaktív node-okat az egyes elágazásokba, egymásra építve, egy fa-szerű struktúrát kialakítva, ennek az előnye elsősorban az, hogy sokszor, főként, ha a Natron más node-jaival dolgozunk, intuitívebb lehet az elem ilyen módon történő használata. Az ennek a megközelítésnek megfelelő node-gráfot a következő ábra szemlélteti (a forrás videónál látható hibától most tekintsünk el, a forrásfájl már nem volt elérhető, amikor a képet készítettem):



10. ábra Interaktív elemek Natronban, fa típusú node-gráffal

- A másik lehetőség az, hogy az egyes interaktív elemek reprezentáljanak egy-egy külön ágat és azokat a beépített Merge node-ok segítségével adjuk hozzá a videóhoz:



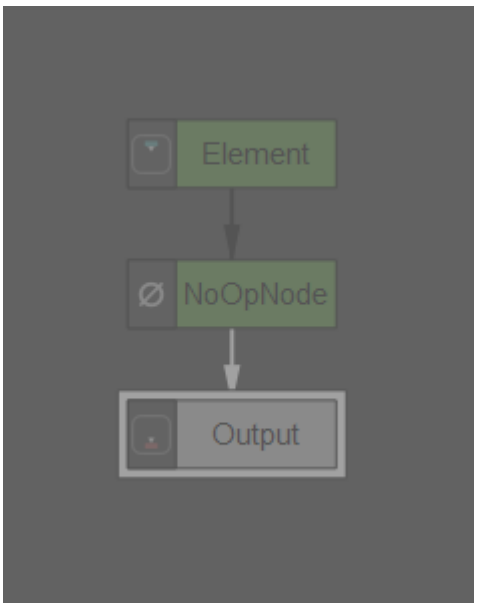
11. ábra Interaktív elemek Natronban, Merge node-ok segítségével

Első megközelítésben a korábbi opciót választottam, hiszen, mint látható, az kevesebb csomópontból áll és sokszor logikusabb is lehet, mint az utóbbi, csakhogy idővel rátaláltam annak a nagy hibájára is: ha az egyes interaktív elemeket időzíteni szeretnénk, akkor az nem fog működni, mivel akkor az adott megjelenítési időtartam nemcsak a kiválasztott elemre, hanem az egész abba érkező forrás ágra is vonatkozni fog. Ez a fajta mellékhatás, ha a művész nem számít rá, roppant kellemetlen tud lenni, így végül mégis a második lehetőség alapján készítettem el a saját implementációmat.

6.4.3.2 Saját node

Az interaktív funkcionalitást támogató szerkesztő bővítmény elkészítésének első lépése egy saját node típus létrehozása, amely egy külön erre a célra létrehozott PyPlug lett.

Amit még kezdés előtt tudni érdemes, hogy Natronban az egyedi node-ok valójában node-csoportok, amiket kívülről egynek látunk. Jelen esetben, mivel maga a csomópont csak adminisztratív célokat látna el, elég lenne egy bemeneti (Input) és egy kimenetei (Output) node is, azonban, mint kiderült a Natron nem engedi az Inputot közvetlen az Outputhoz kötni, így be kellett iktatnom egy köztes NoOpNode-ot is, ami tulajdonképpen semmit sem csinál, csak átengedi magán a bemenetet. A szerkesztőnek van egy eszköze, amivel a kódból létrehozott node-csoportot kirajzolja, azzal megjelenítve mindez az alábbi módon néz ki:



12. ábra Interaktív Natron node csoport belső szerkezete

Ezután már csak a megfelelő inputok hozzáadása szükséges a node tulajdonságai panelhez, ami gyakorlatilag három mezőt jelent:

- **Az interakció típusa:** ez egy dropdown, ahol az 5.2 fejezetben ismertetett interakciós típusok között válogathatunk.
- **Az interakció célja:** szöveges mező, annak következtében, hogy ez a típustól függően sokféle értéket felvehet (pl. ugrási időpont vagy hiperhivatkozás).
- **Külső adatforrás elérhetősége:** ez egy opcionális, szöveges mező, aminek csak akkor van szerepe, ha az interakció típusa cserélhető tartalom lenne, ahol ez a mező adná meg a külső adatforrást. Az értéke tipikusan egy fájl név, útvonal, vagy hiperhivatkozás.

Ezzel a node el is készült, a következő lépés az exportálása.

6.4.3.3 Adatok exportálása

Az adatok exportálását, ahogy már korábban is említettem, egyrészt a streamben kódolt interakciós adatok esetén, másrészt külön fájlban is biztosítanom kellett. Erre a problémára az alábbi megoldással álltam elő: mivel a korábban elkészített enkóder bemenete is egy szöveges fájl, átalakítottam úgy a beolvasást, hogy ugyanabban a formátumban várja az adatokat, mint a webes lejátszó, így az egyetlen dolog, amit a szerkesztőben tenni kell, nem más, mint ennek a fájlnak az előállítása, hiszen annak birtokában innentől mindkét típusú interakciós adattárolás támogatottnak tekinthető.

A Natron által biztosított lehetőségeket áttekintve arra jutottam, hogy az exportálás procedúra elindítása a menüből lenne a leglogikusabb és a legkényelmesebb, így létrehoztam neki egy külön menüpontot. Mivel a menüpontokat a PyPlugoktól eltérő módon lehet scriptelni, így erre egy külön python scriptet volt muszáj létrehoznom, amit teljesen máshol kell tárolni, illetve kicsit máshogy is szükséges kezelni, így lényegében mondhatjuk, hogy maga a bővítmény valójában két pluginból áll, még ha egyébként egynek is tekinthető.

Magának az exportáló scriptnek a működése az alábbi fő lépésekre bontható:

1. Új menüpont beregisztrálása.
2. A menüpontra történő kattintás esetén egy dialógus létrehozása, ahol bekérem az exportálandó fájl útvonalát.
3. Az útvonal kiválasztása után az exportálás elindítása az összes node összegyűjtésével, majd ezekből az interaktívak kiszűrésével.
4. A kiszűrt node-ok végigiterálása és azok adatainak összegyűjtése.
5. A node-ok interakciós típusának, céljának és időtartamának, valamint a bemeneti csomópontjaik tartalmának és pozíciójának koordinátáinak, illetve adott esettben a külső adatforrás elérhetőségének kiírása a kimeneti fájlba. Itt van egy olyan apró bonyodalom, hogy a Natron az egyes képkockák megjelenési idejét frame-ekben írja le, így azt át kell alakítanunk időre, szerencsére a videó kimeneti frame rátáját ismerjük, így ez viszonylag könnyedén kiszámolható a frame számának ezzel történő osztásával. Annak következtében, hogy most már az enkóder is ugyanezt a fájlt kapja meg, ott ezt vissza kell alakítanunk frame-ekre, hogy megkapjuk a megfelelő POC értékeket, de mivel ott is ismerjük az enkódolási frame rátát, ez megtehető, mindössze arra kell figyelnünk, hogy az megegyezzen a szerkesztő által használt értékkel.

7 Értékelés

A rendszer implementációjának befejezése után nem feledkezhetünk meg annak értékeléséről sem, hiszen ez az, ahol kiderül, hogy igazából milyen minőségű szoftvert is sikerült elkészíteni. A következőkben az alábbi szempontok szerint igyekszem a kapott megoldásokat megvizsgálni:

- **Funkcionalitás:** a célként kitűzött funkcionalitást milyen szinten sikerült elérni? Mi az, ami esetleg kimaradt?
    - **Interakciók támogatása:** mennyire sikerült szélesre a támogatott interakciók skálája? Mi az, amit ezekkel el lehet érni.?
- **Teljesítmény:** az elkészült rendszer különböző erőforráshasználati statisztikák alapján mennyire lett hatékony?
    - **Tömörítés hatékonysága:** milyen mértékű tömörítési rátát biztosít a rendszer számunkra, avagy az eredeti adatokhoz viszonyítva mennyivel növeli a videók tárhelyszükségletét az interaktivitás hozzáadása?
    - **Enkóder teljesítménye:** mekkora teljesítmény csökkenéssel jár, ha az általam módosított enkóderrel készítünk videófájlokat, annak eredeti változatárhoz képest?
    - **Dekóder teljesítménye:** a teljes videót direkt módon a vizsgált dekóderen átengedve, mennyivel nő annak a futási ideje a referencia értékek viszonylatában?
    - **Lejátszás sebessége, pufferelés:** ezek sokkal nehezebben mérhető dolgok. Itt alapvetően azt nézem meg, hogy mennyire folyamatos a videó, előfordulnak-e apróbb elakadások az egyes ugrások, valamint elágazások környékén.
- **Használhatóság:** saját véleményem szerint egy átlagos felhasználó számára mennyire tekinthető intuitívnek, illetve kényelmesnek az elkészített szoftverek használata? Mi az, ami nem elég jó?

7.1 Funkcionalitás

Az elkészült szoftverekkel funkcionális szempontból alapvetően elégedett vagyok, gyakorlatilag egy interaktív videók lejátszására képes rendszer összes főbb komponensét sikerült megvizsgálni és azokba valamilyen formájú interaktivitási támogatást beépíteni. Természetesen, mint az általában lenni szokott, az idő előrehaladtával újabb és újabb remek ötletek merülnek fel, amikkel a rendszer egészén javítani lehetne, azonban a rendelkezésre álló időkeret véges, így talán az egyetlen negatívum, amit itt meg tudnék említeni az, hogy ezek közül nem sikerült még többet megvalósítani, de, mint említettem összességében elégedett vagyok. Mindent összevetve, rengeteg különböző részletet megvizsgáltam és számos ideát sikerrel implementáltam.

7.1.1 Interakciók támogatása

A támogatott interakciók részletekbe menőségének vizsgálatához először is tekintsük át még egyszer, hogy eddig milyen interakciós típusokról esett szó és azok mire használhatók:

- **Egyszerű tartalom:** lényegében mindenre jó, amit HTML-lel is meg tudunk valósítani, például akár a 3. fejezetben említett interaktív űrlapok elkészítésére is használható.
- **Egyszerű ugrás:** lehetővé teszi, hogy a videón belül felhasználói interakcióra lényegében bárhonnan bárhová ugorjunk.
- **Automatikus ugrás:** időben párhuzamosan futó branch-ek egyesítésére használható.
- **Elágazás:** a történetekben megjelenő nagyobb döntések, választások fő alkotóeleme.
- **Loopolt elágazás:** ugyanaz, mint az elágazás, de ahelyett, hogy a lejátszás megállna, egy loopolt videószakasz kerül újra és újra megjelenítésre.
- **Hiperhivatkozás:** egy egyszerű hiperhivatkozás, tulajdonképpen HTML-lel is elkészíthető lenne, de ez egy elég gyakran, sokoldalúan alkalmazható elem, így mégiscsak külön típust érdemelt.

- **Interaktív kártya:** egy praktikus elem, inkább kozmetikai célokat szolgál, egyébként ez is megvalósítható lenne egy szimpla tartalom elemmel, igaz elég körülményesen.
- **Cserélhető tartalom külső adatforrással:** ezzel külső forrásokból érkező tartalmat is tudunk kezelni, ami egy rendkívül jelentős bővítése az eddigi funkcióknak, hiszen ennek megváltoztatásához nem kell magához a videóhoz hozzányúlni.

Ha összességében végigtekintek az elkészült interakciós típusokon, akkor ezekkel a legtöbb általam elképzelt lehetséges use-case-t le lehet fedni. Mindössze két, elég speciális hiányzó elem jutott mostanáig eszembe: egy olyan komponens, amivel interaktív videókat lehetne megjeleníteni egy interaktív videón belül, lehetőleg külső forrásból, tehát egymástól teljesen függetlenül, valamint egy olyan elem, ami 3D objektumok, esetleg hosszabb távon komplexebb 3D terek megjelenítésére is képes lenne a videón belül úgy, hogy közben magukkal a háromdimenziós objektumokkal is interakcióba léphetünk.

7.2 Teljesítmény

A teljesítmény mutatók gyakorlatilag minden szoftver értékelésének jelentős szempontjai, hiszen ahogy haladunk az egyre erősebb hardverek felé, a felhasználók elvárásai is úgy nőnek, mind a minőség, mind a válaszidők terén. Ebben a fejezetben különböző mérések alapján értékelem az elkészült rendszert, annak teljesítményére fókuszálva.

7.2.1 Enkóder teljesítménye

Ebben a részben az enkóder kódolási teljesítményt fogom vizsgálni, az ahhoz szükséges idő szempontjából.

A metodológia a következő: ugyanazzal a nyers yuv fájllal futtatom az enkódert először a hivatalos repositoryból is elérhető x265 változattal (3.5 verzió) [13], majd annak az általam interakciók támogatásával kiegészített variációjával. Mindkét tesztet az eredeti futtatáson felül még kétszer megismételve, csökkentve ezzel a mérési hiba esélyét.

Az enkódolás legfontosabb paraméterei a következők: egy 500 frame-ből álló 1920x1080 felbontású képfolyamot szeretnék 60 fps-sel, x265 enkóderrel (illetve annak

a módosított változatával) egy H.265 szabványt követő .mp4 videó fájllá kódolni. Az enkóder minden esetben 8 szálat használ.

Az eredmények (zárójelben az egyes időtartamok mellet látható a kódolás átlagos sebessége is):

Mérés	**Kódolás időtartama (referencia)**	**Kódolás időtartama (módosított verzió)**
1. mérés	84.26s (5.93 fps)	89.01s (5.62 fps)
2. mérés	89.54s (5.58 fps)	85.59s (5.84 fps)
3. mérés	90.59s (5.52 fps)	83.95s (5.96 fps)
Átlag	88,13s	86,18s

Az eredményekből az látszik, hogy az értékek nagyjából azonosak mindkét esetben, sőt átlagban az interaktivitásokkal bővített verzió jobban produkált, ennek oka valószínűleg a teszteléshez használt számítógép aktuális terheltségéből adódik, viszont abból, hogy az enkóder mellett futó, egyébként nem nagy teljesítményigényű alkalmazások ily módon tudták befolyásolni az eredményeket - amik egyénileg egyébként sem voltak jelentősen eltérőek egymástól, egyik esetben sem – arra következtethetünk, hogy az interaktív funkciók támogatása nem ad jelentősebb overheadet a kódolás idejéhez, mi több annak költsége a mérések alapján gyakorlatilag elhanyagolhatónak tűnik.

7.2.2 Dekóder teljesítménye

A dekóder vizsgálata és annak körülményei gyakorlatilag teljesen megegyeznek az enkóderével, ugyanazt a videót is használtam mindkét esetben, azzal az apró különbséggel, hogy most a kódolás pont inverz irányban történik és sajnos a dekódert csak egy szálon tudtam futtatni.

Eredmények (a dekóder kizárólag fps statisztikát adott vissza, így én is azt preferálom, de zárójelben kiszámoltam mellékük azt is, hogy ez egyébként mennyi időt jelent a bemeneti videóra):

Mérés	Dekódolás sebessége (referencia)	Dekódolás sebessége (módosított verzió)
1. mérés	4,23fps (118,20s)	4,17fps (119,90s)
2. mérés	4,29fps (116,55s)	4,16fps (120,19s)
3. mérés	4,22fps(118,48s)	4,20fps (119,05s)
Átlag	4,247fps	4,177fps

Az eredményekből egy enyhe teljesítménybeli csökkenés megfigyelhető a módosított esetben, de ez a különbség itt is közel elhanyagolható.

7.2.3 Tömörítés hatékonysága

Ezen fejezetben az interaktív videók tárhelyigényét hasonlítom azok standard változatához, megkülönböztetve azon eseteket, ha az interaktivitási információkat a streamben vagy külön fájlban tároljuk.

7.2.3.1 Stream-ben tárolt információk

A fájlok méretét a következők szerint fogom tanulmányozni, veszek néhány binárisan tárolt képsorozatot, ezeket az előző fejezetekhez hasonlóan kódolom az enkóder hivatalos változatával, majd az általam készített implementáció segítségével és végül megnézem a különbséget.

Az eredményeket az alábbi táblázat foglalja össze, itt elég sok különféle esetet meg lehetne vizsgálni, az egyszerűség kedvéért most egyetlen interakció hozzáadása mellett néztem az eredményeket:

Videó	Bináris mérete (bájt)	Referencia méret	Interaktív videó mérete	Növekedés mértéke
cactus (1920x1080)	1 555 200 000	4 073 097	4 073 107	10 bájt, 0,00005%
flower (352x288)	37 127	663 290	663 302	12 bájt, 0,002%
bridge_far (352x288)	319 499 114	422 216	422 228	12 bájt, 0,003%

Látható, hogy egyetlen interakció esetén ez az érték elenyésző, ~10 bájtos többet körül mozog, ebből már következtethetünk arra is, hogy körülbelül hogyan alakulhat ugyanez több interakció esetén.

7.2.3.2 Külön fájl

Külön fájl esetén sokkal egyszerűbb a dolog, egyrészt adott a videó mérete, ez minden esetben ugyanannyi, másrészt a külső fájl mérete. Itt két különböző interakcióra néztem meg, hogy egyenként mekkora helyett foglalnak el, az egyik esetben ez az érték 24 bájt, a másik esetén pedig 20 bájt lett. Külső fájl esetén még csak tömörítés sincsen, így innentől a fájl mérete lineárisan skálázódik minden egyes további interakcióra.

7.2.4 Lejátszás sebessége, pufferelés

Sajnos itt konkrét számszerű adatokkal nem tudtam előállni, a minőséget inkább megérzés alapján tudtam vizsgálni. Legalább azt, hogy egyáltalán működik-e a hálózati pufferelés elágazások esetén, sikerült tesztelni, ehhez a módszerem a következő: feltöltöttem a webes videólejátszómat egy valódi webszerverre és megnyitottam annak a távoli gépen futó példányát, amikor elértem egy elágazáshoz és ott a lejátszás megállt, megszakítottam az internetkapcsolatot, majd választottam egy tovább haladási irányt, a lejátszás legrosszabb esetben is még 1-2 másodpercig folytatódott, ebből arra következtettem, hogy a hálózati pufferelés az elvárásoknak megfelelően működik.

Emellett érzés alapján azt tudtam vizsgálni, hogy mennyire folyamatos elágazások esetén a videók lejátszása. Többszöri próbálkozás után, különböző hosszúságú videók esetén, mindössze egyetlen alkalom volt, amikor kicsit úgy éreztem, mintha megakadt volna egy töredék másodperce, de ez persze lehetett csak a véletlen műve is. Összességében a pufferelési megoldásom, saját szubjektív véleményem szerint elég hatékonynak tekinthető.

7.3 Használhatóság

Használhatóság szempontjából az elmondható, hogy a rendszer használható, ellenben kényelem terén még bőven van tér a fejlődésre, hogy csak néhány példát említsek:

- A videószerkesztőben az ugrási célok definiálása jelenleg egy input mező segítségével zajlik, sokkal felhasználóbarátabb lenne, ha ehhez kihasználhatnánk a Natron által nyújtott idővonalat, például úgy, hogy amíg a beviteli mező aktív, addig az idővonalon történő kattintások alapján frissítjük annak az értékét.
- A webes lejátszó felülete funkcionálisan megfelelő, de nem szép, messze lemarad a modern videólejátszókétól.
- A seeker használata lehetne kényelmesebb is, erre egy ötletet a 5.6.2 fejezetben mutattam be.
- A videólejátszó és az azokhoz tartozó pluginok telepítése könnyen összezavarhat egy tapasztalatlanabb felhasználót, annak következtében, hogy külön telepítőt nem készítettem hozzá, jelenleg az installációs procedúra több fájl másolásából/mozgatásából áll különböző könyvtárak között.

Ezekkel a kisebb-nagyobb kényelmetlenségekkel jelenleg nincs is probléma, hiszen a rendszer inkább demonstrációs célokat szolgál, de ezek olyan apróságok, amikkel hosszabb távon mindenképpen szeretnék még foglalkozni és amik miatt az elkészült alkalmazások használatát egyelőre nem nevezném komfortosnak. Ha röviden szeretném jellemezni az eddig felsorolt szoftverek egészét, akkor azt hiszem a fejezet első mondata tökéletesen jól körülírja az aktuális szituációt: „a rendszer használható, ellenben kényelem terén még bőven van tér a fejlődésre”.

7.4 Összegzés

Összegezve azt kell mondjam elégedett vagyok az elkészült rendszerrel, sikerült egy megfelelően hatékony, komolyabb teljesítménybeli overhead nélkül operáló, nem olyan jelentős tárhelytöbbletet igénylő megoldást, azaz megoldásokat létrehozni interaktív videók lejátszásához, és azt kell mondjam egyes számokon, mint például az enkóder mérési eredményei, még én is kifejezetten pozitívan meglepődtem. Használhatóság terén még van hová fejlődni, de remélhetőleg a jövőben annak javítására is sikerül majd sort kerítenem.

8 Továbbfejlesztési lehetőségek

Bár az eddig ismertetett komponenseket sikeresen elkészítettem, azokon bőven akadnak még olyan dolgok, amiken lehetne javítani, valamint olyan funkciók és komponensek, amikkel a rendszer egészét lehetne bővíteni, néhány ötlet ezekre:

- A vizsgált és elkészített komponenseket egyetlen, közös rendszerbe összehozni, áthidalva az egyes platformok közötti különbségeket és azok sajátosságait, tehát például megoldani azt, hogy a streamben kódolt adatok a webes lejátszóban is feldolgozhatók legyenek, remélhetőleg a WebCodecs [10] megjelenésével erre lehetőség is nyílik majd.
- Seeker-fa tényleges implementációja.
- Videólejátszó felületének felhasználóbarátabbá alakítása, a felület stílusának modernizálásával és kényelmi funkciók hozzáadásával.
- Videószerkesztő használatának kényelmesebbé tétele.
- Egy asztali lejátszó implementálása a webes párjához hasonlóan, vagy egy már létező lejátszó interaktív funkciókkal történő kiegészítése.
- További potenciális interakciós típusok támogatása (pl. 3D elemek).
- A jelenlegi interakciók javítása (pl. lehessen interaktív videóba a képernyő egy részén egy másik interaktív videót elhelyezni, mint cserélhető tartalom, miközben a két videó kezelése továbbra is egymástól teljesen függetlenül történik).
- Segédszoftverek létrehozása a szoftverek egyszerű telepítése érdekében a végfelhasználók számára (itt elsősorban a szerkesztő pluginokra, esetleg az en/dekóderre gondolok, hiszen a webes lejátszó akár egy nyilvános weboldalon keresztül is elérhetővé tehető).

9 Összefoglalás

Összességében elmondható, hogy az interaktív videólejátszás egy ígéretes, feltörekvő technológia, amely rengeteg potenciált rejt és várhatóan még számtalan év és kiaknázatlan fejlesztési lehetőség áll előttünk mire elérhetünk addig az állapotig, ahol az már megfelelően felhasználóbaráttá válik és akár a mindennapjaink részét is képezheti.

Dolgozatomban végigtekintettem a témakörben jelenleg elérhető lehetőségeket, valamint egy-két már létező megvalósítási gondolatot egy ilyen rendszer létrehozására, majd részletesebben is megvizsgáltam a saját ötleteimet és azok alapján néhány konstrukciót interaktív videók lejátszására. A dolgozat során elindultunk az ötlettől és annak motivációjától, végigtekintettük az interaktív videólejátszás főbb, általános érvényű kérdéseit, végül az általam elkészített, konkrét enkóder, dekóder, videólejátszó, valamint szerkesztő szoftverek implementációjának fontosabb gondolatait is bemutattam, mindezeket pedig a rendszer egészének értékelése zárta le.

Bár a kitűzött célokat úgy gondolom sikeresen elértem, a dolgozat végén számos továbbfejlesztési lehetőséget is megemlítettem, reményeim szerint lesz lehetőségem még dolgozni azokon a jövőben.

Számomra a videók színfalak mögötti működése egy viszonylag új és érdekes témának számított, bár az alapvető fogalmak egy jelentős hányadát ezelőtt is ismertem, valamint a főbb koncepciókról is hallottam már legalább említés szintjén, ilyen részletességgel még sosem tapasztalhattam meg, hogy hogyan is funkcionálnak ezek a rendszerek, így ez egyértelműen egy izgalmas és elég tanulságos „felfedezőtúra” volt számomra.

Irodalomjegyzék

[1] David I. Schwartz: *„Beyond 'Bandersnatch,' the future of interactive TV is bright"*, https://theconversation.com/beyond-bandersnatch-the-future-of-interactive-tv-is-bright-111037 (Megtekintve: 2021. máj. 14.)

[2] Riad I. Hammoud: *Interactive Video - Algorithms and Technologies*, ISBN 978-3-540-33215-2, Springer, Berlin, Heidelberg, 2006

[3] STENZLER, Michael K.; ECKERT, Richard R. *Interactive video*. ACM SIGCHI Bulletin, 1996, 28.2: 76-81.

[4] VERDUGO, Renato, et al. *Interactive films and coconstruction*. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 2011, 7.4: 1-24.

[5] Blue Billywig, *Dina interaktiva videomål är inom räckhåll*, https://www.bluebillywig.com/se/interaktiv-video/ (Megtekintve: 2021. dec. 9.)

[6] HÄKKILÄ, Jonna, et al. *Developing an interactive social cinema concept with a tangible UI.* In: Proceedings of the 13th International Conference on Mobile and Ubiquitous multimedia. 2014. p. 258-259.

[7] ITU-T: *H.265 szabvány specifikáció*, https://www.itu.int/rec/T-REC-H.265 (Megtekintve: 2021. dec. 10.)

[8] Wikipedia: *Network Abstraction Layer*, https://en.wikipedia.org/wiki/Network_Abstraction_Layer (Megtekintve: 2021. máj. 14.)

[9] Allen Sarkisyan: *VideoFrame - HTML5 Video - SMPTE Time Code capturing and Frame Seeking API - Version: 0.2.2*, https://github.com/allensarkisyan/VideoFrame (Megtekintve: 2021. nov. 29.)

[10] W3C: *WebCodecs API Documentation Draft*, https://w3c.github.io/webcodecs (Megtekintve: 2021. dec. 1.)

[11] Unity dokumentáció: *VideoPlayer*, https://docs.unity3d.com/ScriptReference/Video.VideoPlayer.html (Megtekintve: 2021. nov. 26.)

[12] WhatWG: *HTML Media Documentation*, https://html.spec.whatwg.org/multipage/media.html (Megtekintve: 2021. dec. 7.)

[13] VideoLAN: *x265 homepage*, https://www.videolan.org/developers/x265.html (Megtekintve: 2021. máj. 14.)

[14] Struktur AG: *libde265 GitHub page*, https://github.com/strukturag/libde265 (Megtekintve: 2021. máj. 14.)