הטכניון – מכון טכנולוגי לישראל

הפקולטה להנדסת חשמל

Laboratory of Computer Graphics & Multimedia

**נגן ווידאו VR 360°**

חורף תשע"ח

מגישים: חזי בנדה ואלכסיי חרומוב מנחים: אריק אלבהרי ואביאל נחום

תוכן עניינים

[1. מבוא 3](#_Toc509763635)

[2. אתגרים בפיתוח אפליקציות ל- VR 4](#_Toc509763636)

[3. SDKs 5](#_Toc509763637)

[DirectX11 5](#_Toc509763638)

[FFmpeg 5](#_Toc509763639)

[Intel Media SDK 5](#_Toc509763640)

[Oculus PC SDK 6](#_Toc509763641)

[4. מבט על 8](#_Toc509763642)

[ספריות סטטיות 8](#_Toc509763643)

[כלי פיתוח ופרופיל 8](#_Toc509763644)

[5. ארכיטקטורה 9](#_Toc509763645)

[ארגומנטים 9](#_Toc509763646)

[סקירה 9](#_Toc509763647)

[אתחול 10](#_Toc509763648)

[ריצה 11](#_Toc509763649)

[ספרה 12](#_Toc509763650)

[ממשק CHWDevice 15](#_Toc509763651)

[מחלקת CD3D11DeviceOVR 15](#_Toc509763652)

[מחלקת CDecodeD3DRender 16](#_Toc509763653)

[תרשים מחלקות 17](#_Toc509763654)

[אלגוריתם להמרת קצב ריענון 17](#_Toc509763655)

[6. פרופיל ריצה 20](#_Toc509763656)

[Visual Studio Performance Profiler 20](#_Toc509763657)

[Intel VTune Amplifier 21](#_Toc509763658)

[7. סיכום 22](#_Toc509763659)

[8. ביבליוגרפיה 23](#_Toc509763660)

[9. נספחים 23](#_Toc509763661)

[בדיקת האלגוריתם 23](#_Toc509763662)

[10. רשימת איורים 24](#_Toc509763663)

1. מבוא

עולם המציאות המדומה (VR) צובר תאוצה רבה בשנים האחרונות. בשנת 2015 רכשה חברה Facebook את חברת הסטארט-אפ Oculus בסכום של 2 מיליארד דולר. חברה זו פיתחה משקפי מציאות מדומה מתקדמים בשם Oculus Rift. צעד זה נתן דחיפה לתחרות עצומה בענף, בו חברות ענק כמו: Google, HTC, Sony, Microsoft ו- Samsung מפתחות משקפי VR בהשקעה של מאות מיליוני דולרים. בשנת 2014 השיקה גוגל מוצר העשוי מקרטון בלבד שפועל עם כל מכשיר אנדרואיד בעזרת אפליקציה ייעודית. בכך הנגישה גוגל את חווית המציאות המדומה להמונים בעלות נמוכה.

פריצת דרך נוספת בתחום היא השימוש ההולך וגובר בטכנולוגיית צילום ב-360 מעלות. בעזרת טכנולוגיה זו ובשימוש במשקפי מציאות מדומה, יכול המשתמש להרגיש "בתוך" הסרטון, מה שמעצים את חווית המשתמש ונותן לה ממד חדש ומרענן. אלא שחוויה כזאת נתמכת כמעט בלעדית על ידי פלטפורמות YouTube ו- Facebook ומאלצת את המשתמשים להשתמש בהן.

בפרויקט זה אנו ממשיכים את הטרנד של המציאות המדומה על ידי פיתוח נגן ווידאו שיכול לנגן סרטים שצולמו בטכנולוגיית 360 מעלות. לצורך השגת מטרה זו בצורה יעילה, אנו משתפים פעולה עם חברת Intel במטרה לנצל את יכולות ההאצה של המעבדים שלה.

לצורך הוכחת יכולות ראשונית, הנגן מתממשק עם משקפי Oculus Rift, אולם ניתן להרחיב אותו בקלות לתמיכה במשקפיים של חברות נוספות כמו HTC, Microsoft וכו'.

הייחודיות של הנגן שלנו באה לידי ביטוי בשני מישורים:

1. קוד פתוח – ככל הידוע לנו, נכון להיום לא קיים נגן VR אחר המפורסם בקוד פתוח לשימוש של מפתחים ומשתמשי קצה ברישיון חופשי.
2. האצה בחומרה – שיתוף הפעולה עם אינטל מאפשר לנו לפתח נגן יעיל במיוחד, שבו תהליך הפענוח מואץ על ידי מעבדי אינטל מדור 6 ומעלה.

פיתוח אפליקציות למציאות מדומה הינה משימה מאתגרת משום שהיא מציבה קשיים נוספים מעבר לאלו הקיימים בפיתוח משחקי מחשב רגילים. אחד האתגרים המשמעותיים נובע מהצורך להקטין ככל האפשר את ההשהיה בין תזוזות הראש של המשתמש לבין השפעתן על התמונה המוצגת לו. השהייה גדולה מדי עלולה לגרום לתחושת בחילה. בעיה זו הופכת למאתגרת במיוחד כאשר רוצים לנגן סרט שצולם מלכתחילה בקצב ריענון נמוך. כדי להתמודד עם בעיה זו כתבנו אלגוריתם להמרת קצבי ריענון, כך שהנגן יכול להפיק את קצב הריענון הנדרש על ידי שכפול פריימים והצגתם מספר פעמים, מבלי לפגוע בקצב הניגון המקורי של הסרט.

לבסוף נציג פרופיל ריצה אופייני של הנגן ונצביע על צווארי בקבוק והזדמנויות לאופטימיזציה.



Figure 1- Oculus Rift

1. אתגרים בפיתוח אפליקציות ל- VR

פיתוח אפליקציות למציאות מדומה הינה משימה מאתגרת במיוחד. בעוד שהחומרה נדרשת לעמוד בדרישות נוקשות יותר לעומת משחקי מחשב רגילים, גם התוכנה נדרשת לעבור התאמה לעידן ה- VR. נמחיש את הקושי על ידי הדגמת מספר אתגרים ייחודיים בעולם ה- VR:

קצבי ריענון גבוהים יותר: בעוד שמשחקי מחשב רגילים מספקים חוויית משחק איכותית בקצב ריענון של 60Hz, בעולם ה- VR קצב הריענון עומד על 90Hz. כיוון שהמסך כל כך קרוב לעיניים, קצב ריענון נמוך מדי עלול לגרום לאי נוחות, כאב ראש ועוד. תופעה זו ידועה בשם VR Sickness. המשמעות היא שהחומרה נדרשת להפיק מספר גדול יותר של תמונות\שנייה.

רינדור כפול: בעוד שמשחקי מחשב נדרשים לרנדר סצנה מלאה ולהציגה על מסך המחשב, עבור אפליקציות VR צריך לרנדר את אותה הסצנה פעם אחת לכל עין. המשמעות היא שהחומרה נדרשת לרנדר **לפחות** פי 2 פריימים באותו פרק זמן יחסית למה שמצופה במשחקי מחשב רגילים.

זמני תגובה נמוכים יותר: המונח Motion-to-Photon latency נהגה כדי לאמוד את זמן התגובה של קסדת ה- VR. מונח זה מתאר את הזמן שחולף מרגע שהמשתמש מזיז את ראשו ועד הרגע שהתזוזה באה לידי ביטוי בהשתנות הפוטונים על קסדת ה- VR. זמן תגובה ארוך מדי עלול לגרום למשתמשים לחוות VR Sickness. כדי למזער פרמטר זה נדרשת לא רק חומרה עוצמתית, אלא גם התאמה תוכניתית הן מצד האפליקציה והן מצד שירות זמן-הריצה של האפליקציה (Runtime).

רזולוציות גבוהות יותר: קסדת ה- VR מכילה מסך הנמצא קרוב מאוד לעיני המשתמש, אם צפיפות הפיקסלים לא תהיה מספיק גבוהה אז הוא יוכל להבחין בהם, האובייקטים לא ייראו בצורה טבעית וחווית השימוש תיפגע. ככל שהרזולוציה הנדרשת גבוהה יותר צריך להחזיק חומרה חזקה יותר.

שדה ראייה גדול יותר: גורם נוסף שעלול לגרום ל VR Sickness הינו שדה ראייה מצומצם מדי. בני האדם רואים שדה אופקי של כ- 114° ולכן קסדת ה- VR והאפליקציה חייבים לחקות התנהגות זו. המשמעות היא שנכנסים יותר אובייקטים בשדה הראייה ולכן פעולת הרינדור תהיה ארוכה יותר.

רגישות לאיכות: כאשר האפליקציה מנסה לחקות את העולם האמיתי והמסך נמצא כל כך קרוב לעיניים, האובייקטים המוצגים בסצנה נדרשים להיראות טבעיים. המשמעות היא שהחומרה נדרשת לרנדר אובייקטים ברמת דיוק גבוהה יותר מאשר במשחקי מחשב רגילים.

1. SDKs

## DirectX11

ערכת הפיתוח של Microsoft לאפליקציות גרפיות. ערכה זו מאפשרת למפתחים לבצע משימות מולטימדיה כלליות ובפרט פיתוח משחקי מחשב לפלטפורמת חלונות. הערכה מכילה בתוכה סט רחב של APIs שמאפשרים למפתחים לתקשר עם החומרה שמתחת למערכת ההפעלה לצרכי מולטימדיה. בין היתר ניתן למנות את החבילות: Direct3D, DXGI, Direct2D ועוד. בפרויקט שלנו אנחנו משתמשים בעיקר ב- Direct3D (בקיצור: D3D או D3D11), חבילה זו מאפשרת לנו לרנדר ספרה תלת ממדית שעליה נלביש טקסטורות מתחלפות. את חישוב הנקודות על הספרה אנו מבצעים בעזרת ספריית העזר DirectXTK, שעוזרת לבצע את החישוב ביעילות ולפי הדיוק הנדרש.

## FFmpeg

חבילת FFmpeg מכילה כלים חינמיים רבים לטיפול במולטימדיה. בין היתר היא מכילה: ספריות לטיפול בקידוד/פענוח של אודיו/ווידאו, ספריה לריבוב ופיצול תזרימי אודיו/ווידאו ואף נגן שלם. כל הכלים בחבילה ניתנים ברישיון GPL ופועלים בכל מערכות ההפעלה הנפוצות. הנגן שלנו משתמש בחבילה זו על מנת לטפל בפיצול תזרים ווידאו מתוך קונטיינר סטנדרטי.

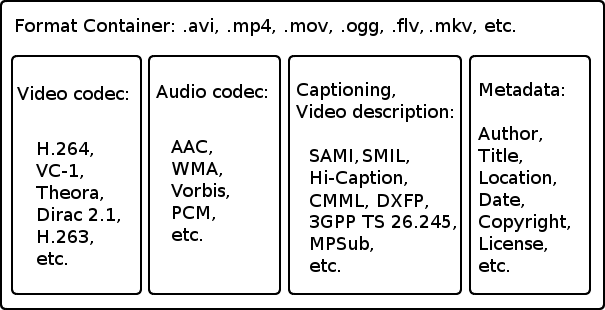


Figure 2- Standard containers

## Intel Media SDK

לאחר שתזרים הווידאו פוצל מתוך הקונטיינר, הוא עדיין מקודד ויש צורך לפענח אותו לפני שניתן להפוך אותו לטקסטורה או להציג אותו על המסך. כדי לבצע משימה זו ביעילות, אנחנו משתמשים במאיץ חומרתי שמגיע עם מעבדים של אינטל מדור 6 ומעלה. חבילת התוכנה שמטפלת במשימות אלה נקראת Media SDK והיא מספקת למפתחים API לביצוע משימות פענוח/קידוד/המרת קידוד יעילים, על מאיץ מיוחד שמותקן במעבד. הנגן שלנו משתמש בחבילה זו לצורך פענוח תזרימי ווידאו בקידוד H264/H265/MPEG. אם לא רוצים להשתמש במאיץ חומרתי אז תמיד אפשר לפענח בתוכנה, אלא שלבחירה זו יש משמעות של פגיעה בביצועים (ולרוב צריכת הספק גדולה יותר), ואילו באפליקציות של מציאות מדומה חשוב מאוד לשמור על השהייה מינימלית.

## Oculus PC SDK

ערכת הפיתוח ל- PC של Oculus מספקת למפתחים את ספריית LibOVR ומספר חבילות נוספות לפיתוח אפליקציות עבור Oculus Rift, ביניהן: Audio SDK, Platform SDK, Avatar SDK. לפיתוח הנגן שלנו אפשר להסתפק בספריית LibOVR, ספריה זו מאפשרת לאפליקציה יכולת להתממשק עם משקפי Oculus Rift ולרנדר עליהם סצנות.

Runtime

תהליך ה Runtime פועל תמיד ברקע בעת ריצת האפליקציה ומספק לה שירותים שעוזרים למשתמש ליהנות מחוויית שימוש איכותית. תהליך זה מקבל מהאפליקציה את התמונה המוכנה להצגה ומבצע עליה עיבוד מקדים בטרם הוא שולח אותה לדרייבר. החלק המשמעותי ביותר של העיבוד המקדים הוא יצירת עיוות מכוון (Distortion) שנועד לפצות על העיוות הפיזיקלי שנוצר על ידי העדשות.

שירות נוסף שנועד להעצים את חווית המשתמש וניתן על ידי ה Runtime נקרא Time Warp, הוא עושה זאת כך:

1. דגימת החיישנים על הקסדה בסמוך ל VSync והכנסת תיקון לתמונה שהתקבלה על ידי האפליקציה, פעולה זו ממזערת את ה Motion-to-Photon latency. זה המצב האידאלי והוא מומחש על ידי האיור הבא.

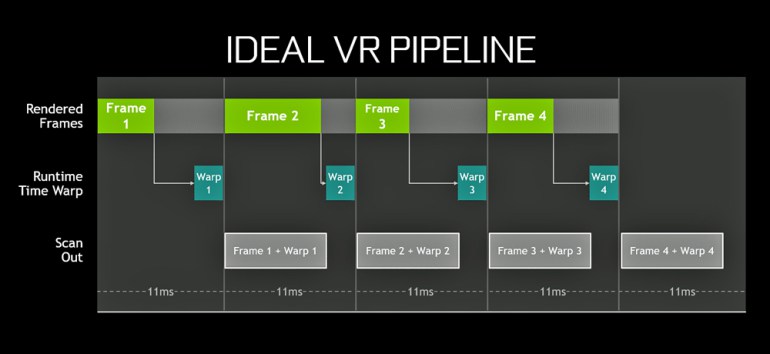


Figure 3 - Ideal VR Pipeline

1. אם האפליקציה לא הספיקה לרנדר תמונה חדשה בזמן, מפעילים את א' על התמונה האחרונה שהתקבלה. מצב זה נקרא Dropped frame והוא מומחש באיור הבא.

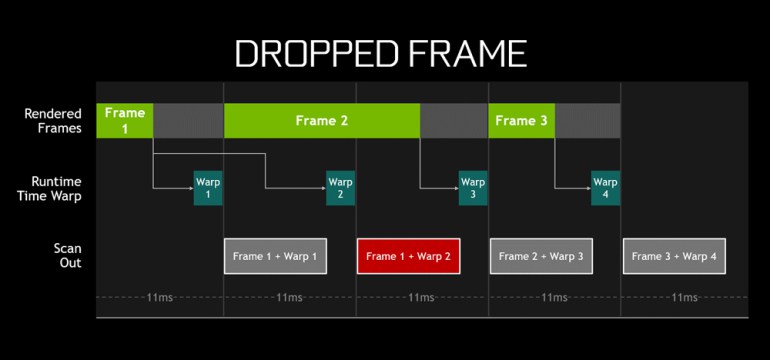


Figure 4 - Dropped Frame

אם שירות ה Time Warp לא מספיק לסיים בזמן לפני VSync, ה- Runtime מציג את התמונה האחרונה שעברה Time Warp. מצב זה נקרא Warp Miss והוא נחשב למצב הגרוע ביותר משום שחוויית המשתמש עלולה להיפגע על ידי ההשהיה שנוצרת במצב זה.

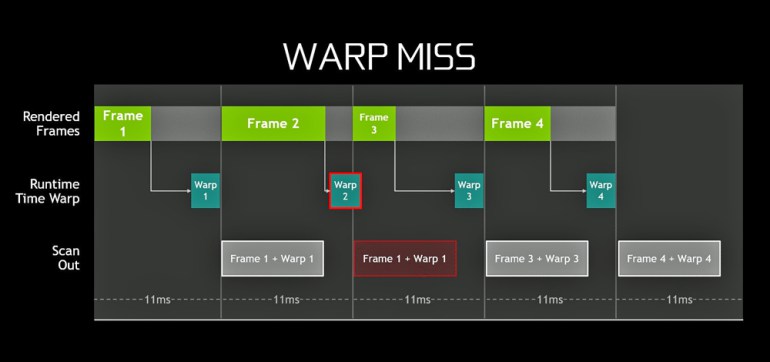


Figure 5 - Warp Miss

1. מבט על

הנגן פותח עבור סביבת Windows 10, ומשתמש לשם כך ב SDK של גרסה זו. על מנת לרנדר אובייקטים על המסך, אנחנו משתמשים ב API של Microsoft לגרפיקה (DirectX11) ובערכת הפיתוח של אינטל Media SDK. בעזרת כלים אלו הנגן יכול לתקשר עם ה- GPU ולהיעזר בו לביצוע חישובים גרפיים. כמו כן הנגן משתמש בערכת הפיתוח החינמית FFmpeg ובעזרתה מתאפשר לו לפצל תזרימי ווידאו מקונטיינרים נפוצים כמו mp4, mkv וכדומה. על מנת לתקשר עם המשקפת של Oculus הנגן משתמש בערכת הפיתוח של Oculus ל- PC.

## ספריות סטטיות

הנגן משתמש ב- 3 ספריות באופן סטטי, נראה את שמותיהם ואת התלויות ביניהן בגרף הבא:



Figure 6- Libraries Dependency Graph

DirectXTK – DirectX Tool Kit

ספריה המתוחזקת על ידי Microsoft ומספקת למשתמשים בה מגוון מחלקות ומבנים נפוצים לאפליקציות DirectX11. אנחנו משתמשים בה בשביל לרנדר ספרה תלת ממדית.

LibOVR

ערכת הפיתוח של Oculus עבור ניהול ההתקשרות עם הקסדה.

sample\_common

המחלקות המשותפות ופונקציות העזר שמגיעות עם ערכת Media SDK, לערכה זו הוספנו את המחלקה התומכת ברינדור D3D11 על קסדת Oculus Rift.

VR360Player

הנגן, מתממשק באופן סטטי לספריות הנ"ל, ובאופן דינמי ל FFmpeg, Media SDK ו- D3D11.

## כלי פיתוח ופרופיל

הנגן פותח בשפת C++ בסביבת Visual Studio 2017. לשם איסוף פרופיל ריצה אופייני בוצע שימוש בכלי הפרופיל המובנה של סביבת הפיתוח וכן בכלי VTune Amplifier של אינטל.

1. ארכיטקטורה

הנגן פותח על בסיס תכנית decode\_sample שמגיעה עם Media SDK, תכנית זו בנויה באופן מודולרי וגמיש מאוד, כך שהכנסת תמיכה במשקפי Oculus Rift לא דרשה שינויים בארכיטקטורת הנגן למעט כתיבת מחלקה ייעודית. כך שגם הוספת תמיכה במשקפיים של חברות אחרות תהיה פעולה פשוטה בזכות הגמישות המובנית בארכיטקטורת הנגן. בפרק זה נתמקד באופן הפעולה של הנגן, קידוד הסרט והצגתו על הספרה מנקודת מבט תוכניתית, נדגיש במיוחד את השינויים שהחלנו על תכנית הבסיס כדי להפוך אותה לנגן גמיש.

## ארגומנטים

כדי להפעיל את הנגן יש לספק לו ארגומנטים במבנה הבא:

<codec id> -ovr -i inputFile

כאשר <codec id> מציין את הקידוד של תזרים הווידאו של inputFile. הוא יכול להיות אחד מאלה:

h264 | h265 | mpeg2

הארגומנט -ovr הינו חובה ומציין שהנגן מרנדר סרט על Oculus Rift. בעתיד יהיו ארגומנטים אחרים שיציינו התממשקות עם משקפיים של חברות אחרות.

הארגומנט inputFIle מציין את קובץ הווידאו שרוצים להפעיל, כאשר תזרים הווידאו שלו מתאים לקידוד שנבחר לעיל.

לדוגמא:

h264 -ovr -i my\_h264\_movie.mp4

## סקירה

האובייקט שאחראי על פעולת הנגן הינו מטיפוס CDecodingPipeline והוא מכיל אובייקטים, משתנים פנימיים ומתודות שעוזרות לו לבצע את משימת הניגון של קובץ הווידאו הניתן לו. ניתן לראות את אופן פעולתו בדיאגרמה הבאה



Figure 7- Player overview

בשלב הראשון הנגן קורא את הארגומנטים שניתנו לו ומפרש אותם. לאחר מכן קוראים למתודת האתחול שמקצה זיכרון לפעולת הפענוח, מאתחלת מבנים פנימיים ויוצרת התקשורת עם ה GPU. כל שגיאה שמתגלה בשלבים אלה נחשבת קריטית ולכן נזנח את פעולת הנגן. לאחר שמתודת האתחול חוזרת, קוראים למתודת הריצה של הנגן. כאן נמצאת הלולאה המרכזית של הנגן, תפקידה לבקש מה- GPU לפענח פריימים ולהציג אותם. בשלב זה יכולות להתגלות מספר שגיאת, אם מתגלה שגיאה ברת התאוששות, ננסה להתגבר עליה על ידי פעולת ריסט לנגן ואם השגיאה לא ברת התאוששות אז נזנח את התכנית ונסיים.

## אתחול

פעולת האתחול מתוארת בתרשים הבא



Figure 8 - CDecodingPipeline::Init()

נשים לב שלכל תיבה בתרשים מתאים צבע שמצביע על API/SDK שנמצא בשימוש בעת ביצוע הפעולה המתוארת. בכל שלב באתחול שבו מתגלה שגיאה אנו זונחים את פעולת הנגן ומסיימים.

## ריצה

פעולת הריצה מתוארת בתרשים הבא



Figure 9 - CDecodingPipeline::Run()

בשלב זה הנגן עובר לפעולת Multithreaded – שני חוטים הפועלים בתצורה של צרכן-יצרן כאשר היצרן הוא החוט הראשי שתפקידו לפענח פריימים והצרכן תפקידו לרנדר את הפריים שהתקבל.

כל הפעולות שצבועות בכחול מתרחשות על ה CPU בלבד ואילו הפעולות הצבועות בירוק מתחילות ב CPU ומועברות ל GPU. פעולות שאנו מתזמנים לצרכי פרופיל מסומנות עם שעון.

לולאת הפענוח בחוט הראשי שולפת מידע מה Bitstream (אובייקט שמכיל מידע בינארי של תזרים הווידאו המקודד) וממלאת אותו במידע חדש לפי הצורך. בכל פעם שיש מספיק מידע, הוא נשלח לפענוח על ה GPU והמידע מתקבל על פני אובייקט שנקרא משטח (Surface) וממנו אפשר לשלוף את הפריים בתצורת פיקסלים NV12 אלא שפורמט זה לא ניתן להמרה לטקסטורה ולכן אנו מבקשים מה GPU לבצע המרה לתצורת פיקסלים RGBA, פעולה זו מתרחשת בשלב ה VPP.

בכל פעם שהחוט הראשי נתקע ללא משטחים פנויים הוא מבצע פעולת סנכרון של משטחים מוכנים, פעולה זו מעוררת את החוט השני והלה שולף משטח מוכן ומתחיל בפעולת הרינדור. בכל פעם שפריים נשלח לרינדור ייתכן והוא יוצג מספר פעמים לפי אלגוריתם להמרת קצבי רענון. לבסוף המשטח מוחזר לחוט הראשי. בשלב זה החוט המשני עשוי להיכנס למצב שינה עד שהחוט הראשי יעיר אותו שוב, אלא שפעולת הרינדור איטית יותר מפעולת הפענוח ולכן תמיד יהיו משטחים מוכנים בהמתנה.

על מנת להבין יותר טוב את האובייקטים והמחלקות הפועלות בפעולת הרינדור, מצורף תרשים הרצף (Sequence Diagram) של פעולת SyncOutputSurface.



Figure 10 - Sequence Diagram

## ספרה

על מנת שהצופה ירגיש שהוא "בתוך" הסרט, עלינו ליצור מסביבו ספרה תלת ממדית שעליה מלבישים את הפריימים של הסרט. יצירת הספרה מתאפשרת בקלות הודות לספריית DirectXTK, ספריה זו מכילה מגוון רחב של מחלקות עזר לרינדור ב D3D11 כאשר יצירת ספרה תלת ממדית היא אחת מהן. אחד הפרמטרים הנדרשים ליצירת הספרה הוא פרמטר ה- tessellation (ריצוף המישור), ככל שניתן בפרמטר זה מספר גבוה יותר, נקבל ספרה המורכבת ממספר גדול יותר של קדקודים כלומר ספרה "חלקה" יותר. מצד אחד רוצים להציג למשתמש תמונה חלקה ככל האפשר אך מצד שני פרמטר זה גורם לגדילה מעריכית במספר הקדקודים שעוברים בצינור של ה- GPU כך שמספר גדול מדי יגרום לפגיעה בביצועים של הנגן ולכן צריך למצוא פשרה.

עבור tessellation=3 קיבלנו את הספרה הבאה:



Figure 11 - Sphere tessellation=3

מספר הקדקודים שהתקבל היה 277 (זניח).

עבור tessellation=4 קיבלנו את הספרה הבאה:



Figure 12 - Sphere tessellation=4

מספר הקדקודים שהתקבל היה 1061 (מעט מאוד).

עבור tessellation=5 קיבלנו את הספרה הבאה:



Figure 13 - Sphere tessellation=5

מספר הקדקודים שהתקבל היה 4165 (מעט).

עבור tessellation=6 קיבלנו את הספרה הבאה:



Figure 14 - Sphere tessellation=6

מספר הקדקודים שהתקבל היה 16517 (בינוני).

הספרה שהתקבלה עבור tessellation=5 הייתה מספקת הן בנראות והן מבחינת מספר קדקודים ולכן בחרנו בפרמטר זה, אך כמובן שהוא ניתן לשינוי.

## ממשק CHWDevice

כאשר רוצים להרחיב את הנגן לתמיכה במשקפיים של חברות אחרות, צריך למעשה לממש את ממשק CHWDevice. ממשק זה מגדיר למעשה חוזה שמחלקות חדשות צריכות לקיים אם הן מעוניינות לספק תמיכה בקסדה שונה (WinMR/HTC/etc).

הממשק נמצא בספריית sampe\_common.

class CHWDevice

{

public:

virtual ~CHWDevice(){}

/\*\* Initializes device for requested processing.

@param[in] hWindow Window handle to bundle device to.

@param[in] srcFPS frame rate of input file. \*/

virtual mfxStatus Init(mfxHDL hWindow, size\_t srcFPS) = 0;

virtual mfxStatus Reset() = 0;

// Get handle can be used for MFX session SetHandle calls

virtual mfxStatus GetHandle(mfxHandleType type, mfxHDL \*pHdl) = 0;

/\*\* Set handle.

Particular device implementation may require other objects to operate. \*/

virtual mfxStatus SetHandle(mfxHandleType type, mfxHDL hdl) = 0;

virtual mfxStatus RenderFrame(mfxFrameSurface1 \* pSurface, mfxFrameAllocator \* pmfxAlloc) = 0;

virtual void Close() = 0;

};

## מחלקת CD3D11DeviceOVR

זוהי המחלקה החדשה שהנגן מוסיף לספריית sample\_common. מחלקה זו אחראית על רינדור פריימים על קסדת Oculus Rift. היא ממלאת את החוזה שמכתיב לה ממשק CHWDevice בעזרת D3D11 וערכת הפיתוח של Oculus, כאשר האחרון אחראי על ניהול ההתקשרות עם הקסדה.

class CD3D11DeviceOVR : public CHWDevice

{

public:

CD3D11DeviceOVR();

virtual ~CD3D11DeviceOVR();

virtual mfxStatus Init(mfxHDL hWindow, size\_t srcFPS);

virtual mfxStatus Reset();

virtual mfxStatus GetHandle(mfxHandleType, mfxHDL \*);

virtual mfxStatus SetHandle(mfxHandleType, mfxHDL);

virtual mfxStatus RenderFrame(mfxFrameSurface1 \*, mfxFrameAllocator \*);

virtual void Close();

virtual void OnKey(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam);

private:

void RenderOVR();

bool InitDevice(int, int, const LUID\*);

void SetAndClearRenderTarget(ID3D11RenderTargetView\*, float, float, float, float);

void SetViewport(float, float, float, float);

void ReleaseDevice();

// \*\*\* private class members \*\*\* //

};

פעולת האתחול שומרת את המצביע לחלון ואת קצב הריענון של קובץ המקור. לאחר מכן מאתחלים את LibOVR ויוצרים שיחה עם הקסדה. בעזרת D3D11 יוצרים Device ו- Context ומאתחלים אותם ואת יתר המשתנים הרלוונטיים. לאחר מכן יוצרים ספרה ומאתחלים מצלמה במרכזה ולבסוף יוצרים תמונת-מראה על המסך.

פעולת הרינדור יוצרת טקסטורה מתוך הפריים המוכן ומקבלת את מיקום הקסדה על פי החיישנים שמותקנים עליה על מנת לחשב את זווית המצלמה בפריים החדש, לצורך חישוב זה נלקח בחשבון גם לחיצה על מקשי המקלדת שמשפיעים על מיקום וזווית המצלמה. לאחר מכן הטקסטורה מולבשת על הספרה ומרונדרת לכל אחת מהעיניים ולבסוף מעתיקים את התוצאה לתמונת-המראה שעל המסך. פעולת הרינדור עשויה להתבצע מספר פעמים, בהתאם לקביעת האלגוריתם להמרת קצב ריענון המתואר בהמשך.

מחלקה זו מורחבת פומבית על ידי מתודת OnKey שנקראת בכל פעם שנלחץ מקש במקלדת ומטרתה לשנות את מיקום וזווית המצלמה בפריים על סמך הקשות על המקלדת.

פלט לדוגמה של תהליך הרינדור מופיע בתמונה הבאה



Figure 15 - Output example

## מחלקת CDecodeD3DRender

שייכת לספריית sample\_common. מחלקה זו היא האחראית ליצירת אובייקט של חלון על המסך, שעליו נראה לבסוף את תמונת המראה. המחלקה גם אחראית לנהל אירועים והודעות המגיעות לחלון כמו למשל DESTORY/MAXIMIZE/etc. כאשר אירועים מסוג KEYUP ו- KEYDOWN מועברים לטיפול במחלקת CD3D11DeviceOVR ומטופלים בה על ידי מתודת OnKey.

## תרשים מחלקות

התרשים הבא מציג את המחלקות המתוארת לעיל ואת יחסי הגומלים ביניהן



Figure 16 - Class Diagram

מחלקת CD3D11DeviceOVR מממשת את ממשק CHWDevice. אובייקט של ממשק זה נולד, חי ומת תחת ידיה של מחלקת CDecodingPipeline וכמוהו גם אובייקט של CDecodeD3DRender, שממנו ניתן להגיע למתודות של CHWDevice.

## אלגוריתם להמרת קצב ריענון

קלט: קצב ריענון מקורי של הסרט, קצב ריענון רצוי של הקסדה ופריימים מוכנים להצגה.

פלט רצוי: שכפול אופטימלי של פריימים, כך שיעמדו בקצב הריענון הנדרש.

דוגמה 1: עבור קצב ריענון מקורי של 30 תמונות\שנייה וקצב רצוי של 90 תמונות\שנייה הפלט הרצוי הוא הצגת כל פריים שלוש פעמים.

דוגמה 2: עבור קצב ריענון מקורי של 60 תמונות\שנייה וקצב רצוי של 90 תמונות\שנייה הפלט הרצוי הוא שכפול כל פריים שני, כלומר: ... רגיל, משוכפל, רגיל, משוכפל, רגיל, משוכפל ...

דוגמה 3: עבור קצב ריענון מקורי של 80 תמונות\שנייה וקצב רצוי של 90 תמונות\שנייה הפלט הרצוי הוא שכפול כל פריים שמיני, כלומר: ... רגיל, רגיל, רגיל, רגיל, רגיל, רגיל, רגיל, משוכפל ...

* הערה לדוגמה 3: נשים לב שלא משנה לנו אם הפריים השמיני ממש הוא זה שמשוכפל או כל אחד מהפריימים 1-7 כל עוד מחזוריות השכפול נשארת קבועה ועומדת בקצב הנדרש. כלומר גם הפלט הבא תקין: ... רגיל, רגיל, רגיל, משוכפל, רגיל, רגיל, רגיל, רגיל, ...

דוגמה 4: עבור קצב ריענון מקורי של 40 תמונות\שנייה וקצב רצוי של 90 תמונות\שנייה הפלט הרצוי הוא שכפול כל שלושה פריימים ושילוש הפריים הרביעי, כלומר: ... משוכפל, משוכפל, משוכפל, משולש ...

* הערה לדוגמה 4: נשים לב שלא משנה לנו אם הפריים הרביעי ממש הוא זה שמשולש או כל אחד מהפריימים 1-3 כל עוד מחזוריות השכפול נשארת קבועה ועומדת בקצב הנדרש. כלומר גם הפלט הבא תקין: ... משוכפל, משולש, משוכפל, משוכפל ...

הגדרה – מחזוריות השכפול: התבנית הקצרה ביותר שחוזרת על עצמה בפלט האלגוריתם.

בדוגמה 1, מחזוריות השכפול היא ... רגיל ...

בדוגמה 2, מחזוריות השכפול היא ... רגיל, משוכפל ...

דוגמאות 3 ו- 4 מציגות את מחזוריות השכפול הנכונות.

הגדרה – שכפול אופטימלי: על מנת להציג למשתמש תמונה "חלקה" ככל האפשר, נגדיר שכפול אופטימלי כך: פלט האלגוריתם ייקרא שכפול אופטימלי אם עבור מחזוריות השכפול, ההפרש בין מספר ההכפלות המרבי למספר ההכפלות המזערי הוא לכל היותר אחד.

הרעיון מאחורי הגדרה זו הוא שהפלט הבא לא מספיק טוב: ... רגיל, משוכפל, משולש ...

מכיוון שהיה אפשר להעביר שכפול מפריים 3 לפריים 1 והיינו מקבלים תוצאה "חלקה" יותר שבה כל פריים משוכפל בדיוק פעמיים: ... משוכפל, משוכפל, משוכפל ...

האלגוריתם:

int p = HRR / FR, q = HRR % FR;

int mod\_p, mod\_q = 0;

while(new frame available) {

mod\_p = 1 % p;

while(mod\_p) {

mod\_p = (mod\_p + 1) % p;

<<render frame>>

}

do {

mod\_q = (mod\_q + q) % (HRR – (p-1) \* FR);

<<render frame>>

} while(mod\_q /FR);

}

כאשר HRR מסמן את קצב הריענון הנדרש (HMD Refresh Rate) ו- FR מסמן את קצב הריענון המקורי של הסרט (Frame Rate). ובהנחה שהקצב המקורי לא גדול מהקצב הרצוי.

טענה: פלט האלגוריתם הינו שכפול אופטימלי.

הוכחה:

ראשית נבחין שלולאת ה while הפנימית מתבצעת אותו מספר פעמים בכל איטרציה של לולאת ה while החיצונית ולכן אין לו השפעה על אופטימליות השכפול. נסתכל על לולאת ה do while וננסה להבין כמה פעמים היא מתבצעת. כאשר ברור שהיא תמיד תתבצע לכל הפחות פעם אחת (מהגדרת do while).

עבור מקרה פרטי q=0:

במקרה זה תמיד מתקיים mod\_q=0 ולכן התנאי של לולאת ה do while תמיד 0, כלומר בכל איטרציה של הלולאה החיצונית, משכפלים את הפריים אותו מספר פעמים.

עבור q>0:

הקשר בין p, q, HRR, FR ניתן לייצוג על ידי המשוואה:



קל לראות שמתקיים:



ולכן המשתנה mod\_q יכול לקבל ערכים שלמים בטווח:



ומכאן שהתנאי ללולאת ה do while תהיה בטווח:



לכן התנאי בלולאת ה do while יכול לקבל רק את הערכים 0 או 1. נרצה להוכיח שנחזור על לולאה זו לכל היותר פעם אחת במחזור שכפול, כלומר יש להוכיח שאם התנאי קיבל את הערך 1, אז באיטרציה הבאה הוא יקבל את הערך 0. נסתכל על הערכים המרביים שיכול לקבל המשתנה mod\_q באיטרציות עוקבות:



נשים לב שעבור הערך הנמוך יותר מתקיים:



כלומר עבור ערך זה (וגם כל הקטנים ממנו) התנאי של לולאת ה do while מקבל 0.

עבור הערך הגדול ביותר מתקיים:



כלומר עבור ערך זה התנאי של לולאת ה do while מקבל 1. ובאיטרציה הבאה:



ומכאן שלכל ערך של q, נחזור על לולאת ה do while לכל היותר פעם אחת בכל מחזור שכפול, ומכאן שמספר השכפולים הגדול ביותר יהיה גדול ממספר השכפולים הקטן ביותר באחד לכל היותר, ולכן הפלט של האלגוריתם הוא שכפול אופטימלי. מש"ל.

1. פרופיל ריצה

בפרק זה נציג את ממצאי הפרופיל שאספנו עבור ריצה אופיינית של התכנית. לצורך האיסוף השתמשנו בשני כלים: Performance Profiler המובנה ב Visual Studio 2017 ו VTune Amplifier של אינטל, כאשר היתרון של הכלי האחרון הוא היכולת למדוד משימות ספציפיות.

## Visual Studio Performance Profiler

הפרופיל שאספנו עם כלי זה הציג תמונת מצב חיובית לגבי תפקוד הנגן:

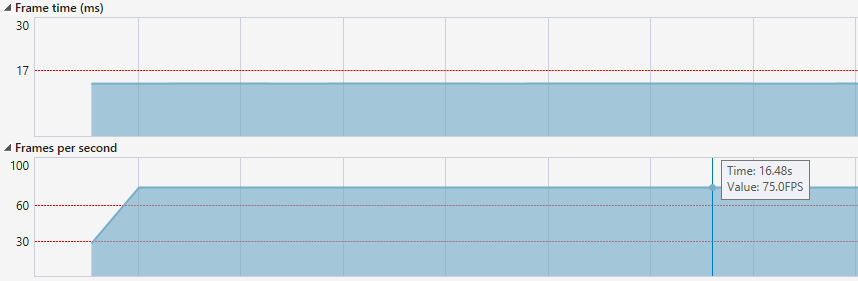


Figure 17 - VS Performance Profiler overview

אפשר לראות שלאחר שלב האתחול של הנגן, הוא מגיע לרוויה ושומר על ביצועים אחידים לאורך כל הסרטון, כאשר קצב הריענון של הסרט הוא קבוע ועומד על 75 fps, בדיוק כמו קצב הריענון של קסדת Oculus DK2. כשמסתכלים על המידע שנאסף על ידי כלי זה ברזולוציה גבוהה יותר, רואים את התמונה האופיינית הבאה:

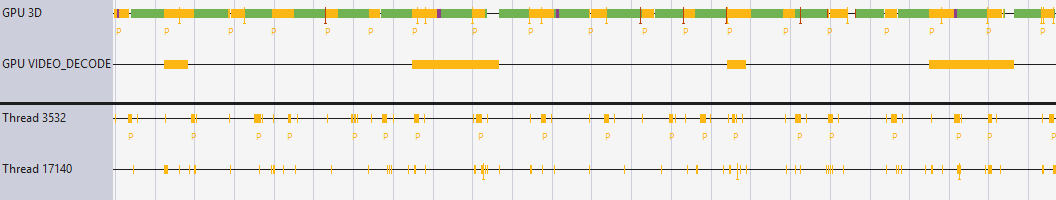


Figure 18- Visual Studio Performance Profiler

הצבע הצהוב מסמן את חלק הזמן שבו הנגן שלנו עבד על ה- GPU ואילו החלק הירוק מסמן את חלק הזמן שבו ה- Runtime של Oculus עבד על ה- GPU. האות p מציינת קריאה ל Present().

מהתבוננות במידע אפשר לראות שמעל מחצית מהזמן שלוקח לרנדר פריים נצרך על ידי שירות ה- Runtime ואילו הנגן שלנו מסיים לרנדר את הפריים בזמן קצר. כדי לאשש טענה זו עיבדנו את הנתונים שנאספו בפרופיל. חלק הזמן שבו פועלת כל קריאה ל- GPU מסוכמת בסכמה הבאה:

Eye 1

Eye 2

Figure 19 - Rendering loop time distribution

עיבוד הנתונים הראה שמרבית הזמן שלוקח לרנדר פריים אכן נצרך על ידי ה- Runtime, כאשר הזמן שלוקח לרנדר את העיוות מסביב לכל עין הוא צוואר הבקבוק המשמעותי ביותר.

## Intel VTune Amplifier

כלי זה ניתן לתכנות על מנת למדוד את אורכן של משימות ספציפיות לאורך ריצת הנגן. לכן ביקשנו למדוד את זמן ביצוע האתחול, את הזמן שלוקח לקרוא מידע חדש לתוך ה Bitstream ואת הפעולות שמתבצעות בלולאת הרינדור על ה- GPU. פרופיל אופייני שנאסף עם כלי זה מוצג בגרף הבא

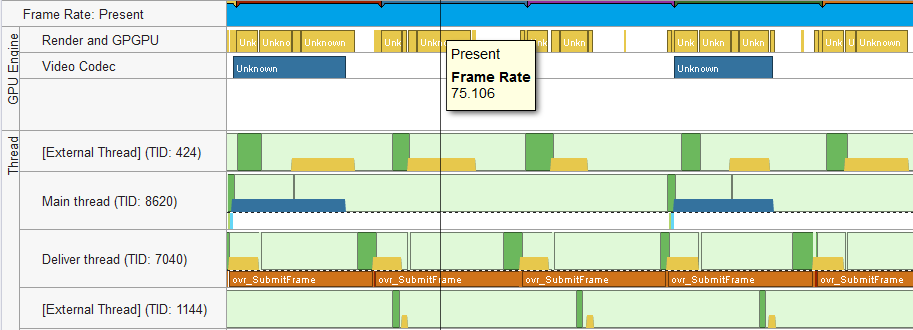


Figure 20 - Intel VTune Amplifier

הסימניות בשורה העליונה מציינות קריאה ל Present(), הצבע הצהוב מציין פעולת רינדור שמתבצעת על ה- GPU, הצבע הכחול מציין פעולת פענוח שמתבצעת על ה- GPU. בצד שמאל מצוין ליד כל חוט מה שמו (Main, Deliver) או שמא הוא שייך לתהליך חיצוני (ה- Runtime).

גם בדיאגרמה זו אפשר לראות שמרבית הזמן שלוקח לרנדר פריים נצרך על ידי ה Runtime כפי שראינו עם הכלי של Microsoft. מצאנו גם שפעולת האתחול של הנגן מסתיימת תוך כ- 0.2 שניות, וכן שמילוי ה Bitstream אורך פחות מ- 1 ms, כלומר פעולות אלו מתבצעות ביעילות ולא מהוות צוואר בקבוק. במבט ראשון נראה שהתוצאות שנאספו עם הכלי הנוכחי תואמות לתוצאות שקיבלנו עם הכלי של Microsoft אלא שעבור רזולוציה נמוכה יותר שמנו לב לתופעה הבאה:

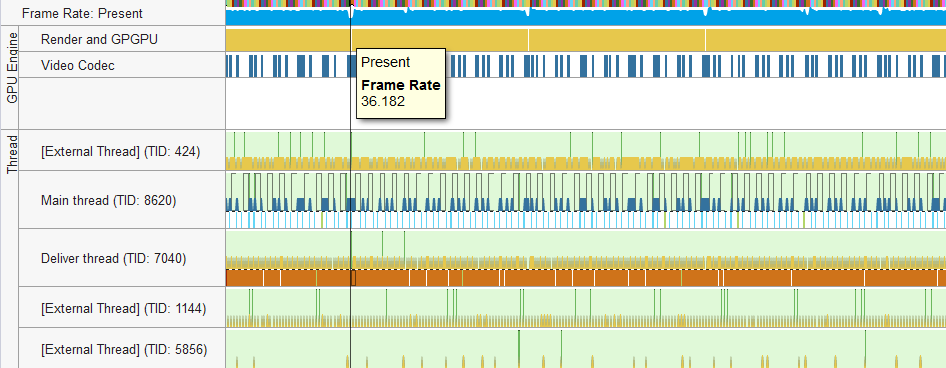


Figure 21 - Performance degradation

אחת לכמה זמן, יש צניחה בביצועים וקצב הריענון נפגע לפרק זמן קצר. תוצאה זו לא נצפתה עם הכלי של Microsoft והיא אף הוחמרה כאשר עברנו מה- Emulator לקסדת ה- DK2. אנחנו מעריכים שהסיבה לצניחה בביצועים נובעת מאחת הסיבות הבאות:

1. הבעיה נגרמת כתוצאה משימוש ב Runtime בגרסתו הישנה (0.8-beta).
2. הבעיה נגרמת כאשר שולחים מספר פריימים לפענוח בזה אחר זה, משאבי ה- GPU נתפסים לצורך פעולת הפענוח ולכן פעולת הרינדור מתעכבת.

בניסיון לשלול את אחת האפשרויות, עברו ל Runtime העדכני ביותר והתאמנו עבורו את ה- API. לצערנו קסדת ה- DK2 לא נתמכת היטב עם ה- Runtime העדכני ולכן לא רק שהנגן לא פעל כשורה, גם אפליקציות פשוטות אחרות קרטעו במצב זה. כדי להתמודד עם המצב החדש, החלפנו את ה- DK2 בקסדה החדשה CV1 אלא שלקסדה זו יש רזולוציה גבוהה יותר ולכן ה- GPU התקשה לספק את הביצועים הנדרשים וגם כאן הנגן שלנו קרטע כמו גם אפליקציות פשוטות יותר. לבסוף החלטנו שנסתפק בלהצביע על הבעיה ועל הכיוונים האפשריים לפתרון כפי שהצענו.

1. סיכום

בפרויקט זה פיתחנו נגן VR שיכול לנגן סרטים שצולמו בטכנולוגיית 360 מעלות. הנגן יודע לקרוא את כל הפורמטים הסטנדרטיים הודות לשימוש ב FFmpeg, לשם המחשת יכולות הכנסנו תמיכה במשקפי Oculus Rift אך המבנה הגמיש של הנגן מאפשר להכניס בקלות תמיכה במשקפיים של כל חברה אחרת. הנגן מריץ שני חוטים בתצורת יצרן-צרכן שפועלים ביעילות ומאותתים זה לזה. על מנת לשפר את חווית המשתמש, כתבנו אלגוריתם להמרת קצב ריענון והוכחנו שהוא אופטימלי. לבסוף, אספנו פרופיל אופייני של הנגן בעזרת שני כלים שונים ומצאנו שה- Runtime של Oculus מהווה את צוואר הבקבוק המשמעותי ביותר, כמו כן מצאנו שמדי פעם יש צניחה בביצועים לפרק זמן קצר, והצבענו על הסיבות האפשריות.

1. ביבליוגרפיה

Intel. (n.d.). *Intel Media Developers Guide.*

Intel. (n.d.). *Media SDK Manual.*

Luna, F. D. (2012). *Introduction to 3D Game Programming with DirectX 11.*

METTLE. (n.d.). Retrieved from https://www.mettle.com/360vr-master-series-free-360-downloads-page/

Microsoft. (n.d.). *DirectXTK Wiki.* Retrieved from Github: https://github.com/Microsoft/DirectXTK/wiki

Oculus. (n.d.). *Introduction to the PC SDK*. Retrieved from https://developer.oculus.com/documentation/pcsdk/latest/concepts/pcsdk-intro/

1. נספחים

## בדיקת האלגוריתם

בעזרת תכנית זו בדקנו את אופן פעולת האלגוריתם להמרת קצב הריענון

#include **<iostream>**#define **HMD\_REFRESH\_RATE** 75  
#define **SRC\_FPS** 30  
  
**void** loop(**int** p, **int** q)  
{  
 **int** mod\_p, mod\_q = 0;  
  
 **int** i = 150;  
 **while**(i--)  
 {  
 mod\_p = 1 % p;  
 std::cout << **"in "**;  
 **while** (mod\_p) {  
 mod\_p = (mod\_p + 1) % p;  
 std::cout << **"loop "**;  
 }  
 **do** {  
 mod\_q = (mod\_q + q) % (HMD\_REFRESH\_RATE - (p - 1) \* SRC\_FPS);  
 std::cout << **"loop"** << **" "**;  
 } **while** (mod\_q / SRC\_FPS);  
 std::cout << **"out"** << std::endl;  
 }  
}  
  
**int** main() {  
 loop(HMD\_REFRESH\_RATE / SRC\_FPS, HMD\_REFRESH\_RATE % SRC\_FPS);  
 **return** 0;  
}

1. רשימת איורים

[Figure 1- Oculus Rift 3](#_Toc509763500)

[Figure 2- Standard containers 5](#_Toc509763501)

[Figure 3 - Ideal VR Pipeline 6](#_Toc509763502)

[Figure 4 - Dropped Frame 6](#_Toc509763503)

[Figure 5 - Warp Miss 7](#_Toc509763504)

[Figure 6- Libraries Dependency Graph 8](#_Toc509763505)

[Figure 7- Player overview 9](#_Toc509763506)

[Figure 8 - CDecodingPipeline::Init() 10](#_Toc509763507)

[Figure 9 - CDecodingPipeline::Run() 11](#_Toc509763508)

[Figure 10 - Sequence Diagram 12](#_Toc509763509)

[Figure 11 - Sphere tessellation=3 13](#_Toc509763510)

[Figure 12 - Sphere tessellation=4 13](#_Toc509763511)

[Figure 13 - Sphere tessellation=5 14](#_Toc509763512)

[Figure 14 - Sphere tessellation=6 14](#_Toc509763513)

[Figure 15 - Output example 16](#_Toc509763514)

[Figure 16 - Class Diagram 17](#_Toc509763515)

[Figure 17 - VS Performance Profiler overview 20](#_Toc509763516)

[Figure 18- Visual Studio Performance Profiler 20](#_Toc509763517)

[Figure 19 - Rendering loop time distribution 21](#_Toc509763518)

[Figure 20 - Intel VTune Amplifier 21](#_Toc509763519)

[Figure 21 - Performance degradation 22](#_Toc509763520)