Universitatea Alexandru Ioan Cuza Iasi
Facultatea de Informatică Iași

LUCRARE DE LICENȚĂ

360 Video Maker

propusă de
Sandu Vlad

Sesiunea: *februarie, 2017*

Coordonator ştiinţific
Conf. Dr. Anca Vitcu

UNIVERSITATEA ALEXANDRU IOAN CUZA IAŞI

FACULTATEA DE INFORMATICĂ

360 Video Maker

Sandu Vlad

Sesiunea: *Iulie, 2017*

Coordonator ştiinţific
Conf. Dr. Anca Vitcu

DECLARAŢIE PRIVIND ORIGINALITATE ŞI RESPECTAREA

DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licenţă cu titlul "*360 Video Maker*" este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învăţământ superior din ţară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe
Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- - toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi deţin referinţa precisă a sursei;
- - reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa precisă;
- - codul sursă, imagini etc. preluate din proiecte *open-source* sau alte surse sunt utilizate cu respectarea drepturilor de autor şi deţin referinţe precise;
- - rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iaşi, 14.02.2017

Absolvent Sandu Vlad

_____

(semnătura în original)

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul "*360 Video Maker*", codul sursă al programelor şi celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea Alexandru Ioan Cuza Iaşi să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iaşi, 14.02.2017

Absolvent Sandu Vlad

_____

(semnătura în original)

ACORD PRIVIND PROPRIETATEA DREPTULUI DE AUTOR

Facultatea de Informatică este de accord ca drepturile de autor asupra programele-calculator, format executabil și sursă, să aparțină autorului prezentei lucrări, *Sandu Vlad.*

Încheierea acestui accord este necesară din următoarele motive:

Doresc să continui dezvoltarea acestui proiect până la punctul în care voi putea să îl distribui.

Iași, 14.02.2017

Decan Adrian Iftene

_____

(semnătura în original)

Absolvent Sandu Vlad

_____

(semnătura în original)

Cuprins

Introducere

Odata cu evolutia tehnologica, camerele video au devenit din ce in ce mai perfomante. Programele de editare si procesare video devin din ce in ce mai cautate iar utilizatorii cer din ce in mai multe functionalitati de la aceastea.

Am observat pe parcursul anilor cum anumite categorii de videouri de pe Youtube incearca sa introduca cat mai multa informatie intr-un singur cadru. Pozele panoramice au devenit din ce in ce mai populare, apoi camerele video cu lentile de tip “Fisheye lens” (Ochi de peste) incearca sa acopere cat mai mult din campul 360 ai privitorului. Abia atunci cand au aparut camerele 360, modul de a vizualiza o imagine sau un video a devenit mult mai interesant si palpitant. Insa aceste camere costa, iar programele de editare al unei astfel de imagine sau video este greu de utilizat.

Odata cu aparitia cu acestor dispovitive numite VR-uri (“Virtual Reality”), am observat ca multi creatori de multimedia isi doresc sa creeze continut special pentru aceste dispozitive, insa programele necesare sunt greu de gasit. Deasemenea, in mall-uri se promoveaza aceste dipozitive, utilizatorul avand acces contra-cost la un video 360. E foarte palpitant, o experienta de neuitat, mai ales atunci cand incepi sa iti pierzi echilibrul. Insa de ce nu am putea urmari un astfel de video, in acelasi stil si acasa, gratis ?

Programul meu incearca sa rezolve aceasta problema. Vreau ca utilizatorul sa isi creeze propiile video-uri 360 simplu si usor. Cu acest program vreau sa le vin in ajutor in special creatorilor de animatii 3D. Acestia pot cu usurinta sa isi randeze video-urile in asa fel incat programul meu sa le poate procesa.

Pornind de la toate aceste probleme, cercetarea a pornit intr-un subdomeniu placut al matematicii, usor de imaginat si vizualizat, intr-o lume a obiectelor tridimensionale, iar simplul fapt ca nu am mai avut de a face cu astfel de lucruri pana acum, m-a facut si mai curios.

Documentatia mea se intinde pe 2 capitole. In primul capitol am prezentat cateva notiuni teoretice despre proiectia unei sfere pe un plan, despre mapa sferica si mapa cubica, si transformarea dintre cele doua. In capitolul 2 am prezentat limbajul de programare folosit, frameworkul, libraria folosita, urmata de descrierea aplicatiei si a modului de utilizare.

Contributii

Programele de procesare si editare video au fost din totdeauna unealta cea mai favorita a creatorilor multimedia. Prin intermediul lor, videoul isi atinge valoarea maxima, din punct de vedere calitativ si cantitativ, telespectatorul vizonand cu placere continutul.

Mereu am fost impresionat de editoarele de fisiere video, cum ar fi After Effects din pachetul Adobe. Modul cum sunt construite, interactiunea cu utilizatorul final, precizia si usurinta acestora, m-a inspirat sa creez si eu soft de editare video, simplu si usor de utilizat.

Soft-urile actuale de editare ale video-urilor 360 le gasesc greoaie pentru utilizatorul final; deobicei vin sub forma unor plugin-uri pentru softuri mai mari, sau programe cu o interfata complicata. Utilizatorul vrea sa ajunga la rezultatul final cat mai repede, sa foloseasca soft-ul de care dispune rapid si simplu. Proiectul meu incearca sa rezolve aceasta problema a utilizatorului, sa ii ofere o interfata simpla de utilizat, sa reduca timpul in procesul de creeare al acestuia.

Am inceput prin a studia cum procedeaza jocurile video atunci cand vine vorba de maparea unei texturi, apoi am continuat cu studiul mai multor proiectii sferice si am ramas la proiectia Equirectangulara, pe care o voi descrie putin mai tarziu.

Programul implementat randeaza un video 360 provenit din 6 video-uri, fiecare avand un camp de vizualire de 90 ° si un raport de 1:1 intre numarul de pixeli pe lungime si latime. Reteaua de socializare Facebook si serviciul video Youtube dispun de controale speciale pentru video-urile 360.

Capitolul I – Tipuri de proiectii 2D

Mapa Cubica si Mapa Sferica

O mapa cubica reprezinta cea mai simpla si eficienta metoda de a stoca mediul sau sky-box-ul dintr-un joc video. Aceasta mapa este trasformata intr-o panorama 360 la runtime.
Ca mod de stocare ne putem imagina ca ne aflam la origine intr-un spatiu tridimensional in interiorul unui cub; apoi luand o camera foto avand un unghi de vizualizare de 90° atat de orizontala cat si pe verticala, indreptam aparatul foto catre cele 6 directii (+x, +y, +z, -x, -y, -z) si facem cate o poza, apoi cele 6 poze le unim ca in figura alaturata, formand mapa cubica.

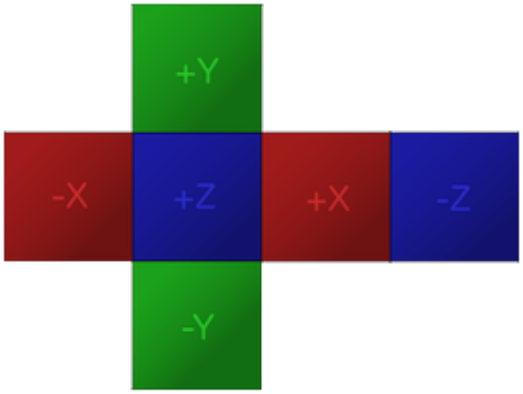


Figura 1 - Mapa cubica

O mapa sferica incearca sa proiecteze cat mai bine suprafata sferei pe un plan. O proiectie sferica perfecta pe un plan 2D este imposibila fara a avea deformari. De aceea, multi geografi si cartografi au venit cu mai multe solutii. Problema unei astfel de proiectii consta in deformarea dimensiunilor si a unghiurilor in zona celor 2 poli.

Spre exemplu proiectia Mercator este o priectie de tip “conformal”, adica isi pastreaza dimensiunea unghiurilor local, insa imaginea se distorsioneaza pe masura ce latitudinea creste.

Figura 2 - Proiectie Mercator

Proiectie Equirectangulara

Proiectia utilizata de mine se numeste “Proiectie equirectangulara”. Modul de construire a acestei proiectii este urmatorul: avand un plan, infasuram sfera cu acesta formand un cilindru; cilindru fiind format din mai multe cercuri suprapuse, trasam un vector din centrul unui cerc pana cand se intersecteaza cu sfera, apoi luam culoarea din acea zona de pe sfera si o punem pe cilindru; la final transformam cilindrul inapoi in plan. Imaginea stocata a unei astfel de proiectii are un raport intre lungime si latime de 2:1. Avantajul cel mai mare ale acestei proiectii, din punct de vedere calitativ, este faptul ca imaginile date ca input nu isi pierd din calitate in urma trasformarii, ba din potriva avem informatie redundanta pe masura ce latitudinea creste, primul si ultimul rand de pixeli fiind formati dintr-o singura culoare.

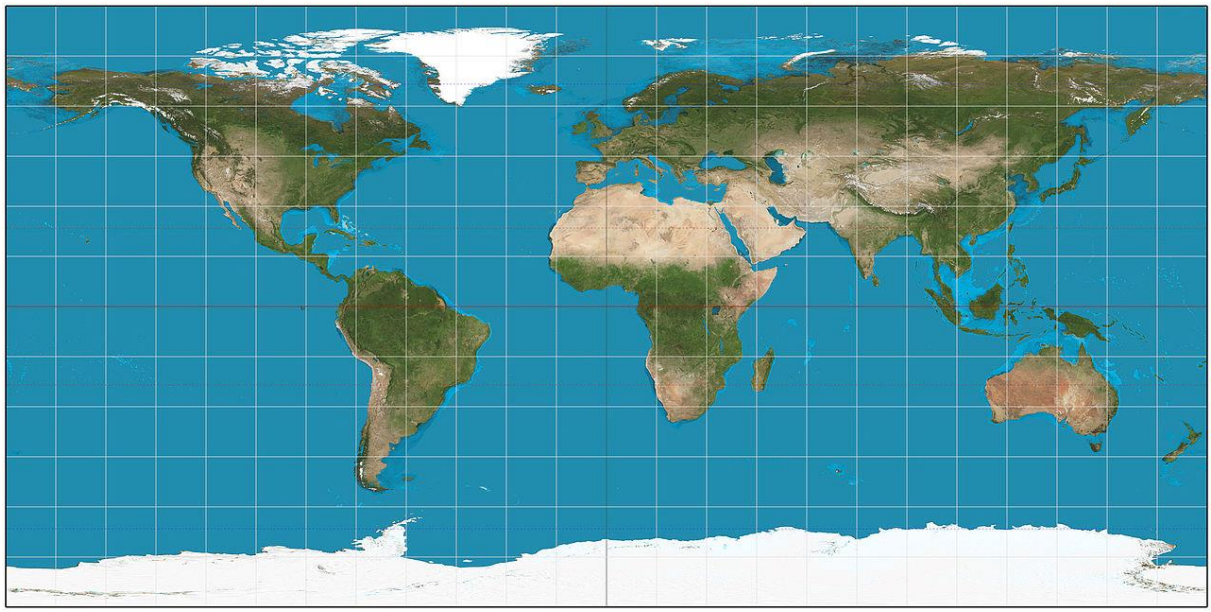


Figura 3 - Proiectie Equirectangulara

Transformarea unei mape cubice intr-o mapa sferica Equirectangulara

Procedeul folosit in program pentru transformarea unei mape cubice intr-o proiectie equirectagulare difera de cel descris mai sus. Reamintim faptul ca datele noastre de input constau in 6 fisiere video care compun o mapa cubica. Ne putem imagina astfel o sfera care se afla in interiorul unui cub. De la fiecare “pixel”/punct de pe cub, trasam un vecter pana la origine. Acest vector se va intersecta la un moment dat cu sfera, iar punctul intersectiei va lua culoarea spefica punctului de pe cub.

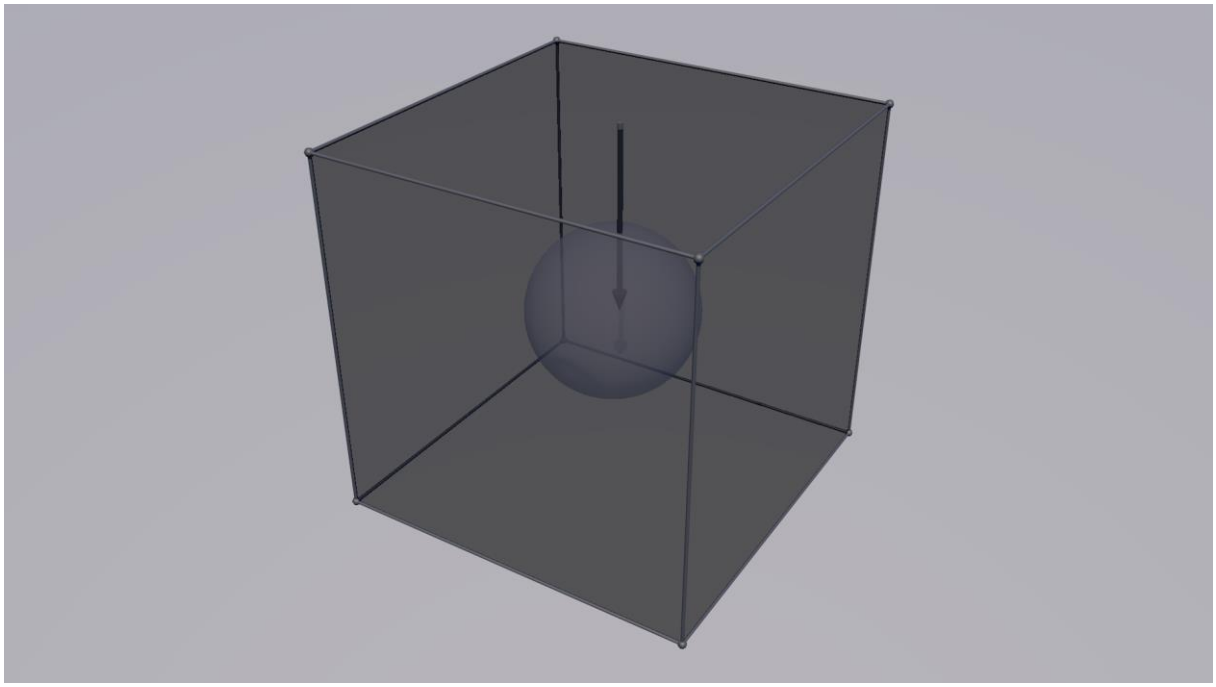


Figura 4 - Vector cu originea pe suprafata cubului ce intersecteaza sfera

Capitolul II – Construirea aplicatiei

Initial ideea aplicatiei a pornit de la ideea urmatore: avand ‘n’ fisiere video, sa se contruiasca o panorama 360. Insa aceasta metoda presupunea un proces complex de lipire a imaginilor (“image stiching”), de aceea am redus problema de la un caz general, la unul particular, si anume, avand 6 fisiere video, sa se construiasca o panorama 360.

Ca mediu de dezvoltare am folosit Visual Studio 2017 Comunity Edition. Aplicatia este una de tip destkop, deoarece se preteaza mult mai bine pentru ceea ce face, utilizatorul putand sa o foloseasca fara conexiune la internet.

Ca arhitectura, am incercat sa elimin cat mai mult coeziunea intre UI si partea de bussiness. Proiectul se bazeaza pe 2 componente importante, PreviewRenderer si RenderEngine.

PreviewRenderer este folosita ca o librarie de proiectul principal. Scopul acestei librarii este acela de a oferi utilizatorului un preview asupra rezultatului final. Am avut grija ca acest preview sa nu consume atat de multe resurse, cum ar fi putere de calcul sau memorie ram.

RenderEngine este ultima librarie care este folosita de aplicatie, rolul ei fiind sa randeze toate frame-urile din fisierele video intr-un singur fisier video.

In figura de mai jos putem observa elementele principale din proiect, fara celelalte librarii folosite. Voi vorbi putin mai tarziu despre fiecare clasa si voi preciza in ce mod este folosita.

In libraria “Entities” am pus clase comune sau “enum”-uri folostie de toate celelalte librarii.

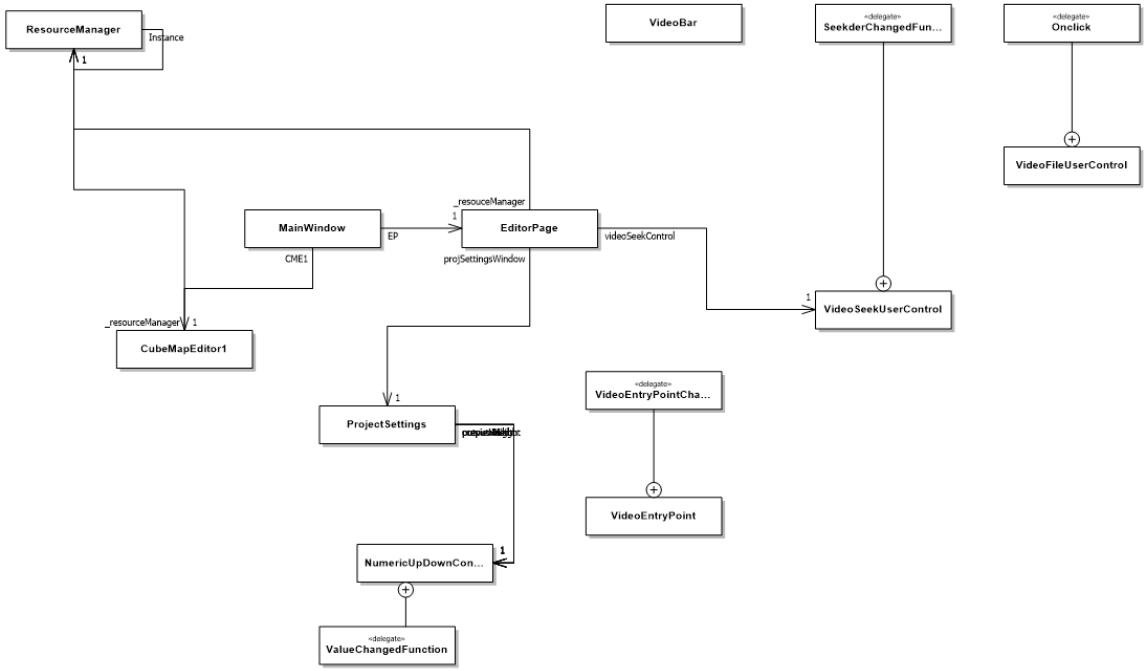


Figura 5 - Arhitectura aplicatiei (doar assembly-ul "360 Video Maker")

De ce C# WPF ?

O Scurta descriere

Am ales sa programez in limbajul C# deoarece il cunosc cel mai bine. Fata de Windows Forms, Windows Presentation Foundation (WPF) ofera un control mult mai bun pentru partea de UI. De la bun inceput am stiut ca voi avea nevoie de controale customizate si de o interfata cat mai flexibila, iar WPF e numai bun pentru asta.

Controale Customizate

WPF imi ofera o lista de controale standard, aceleasi ca si in windows forms, insa prin creearea unui control customizat, am redus codul si complexitatea arhitecturii.

Spre exemplu, WPF nu are controlul numit “numeric up down”. Acest tip de control imi trebuia pentru setarea inaltimii si lungimii imaginilor de preview si randare finala ale video-ului. Fiindca nu am avut o alta optiune, am decis sa imi creez propiul control de tip “numeric up down”. Acest control dispune de cele doua butoane de incrementare si decrementare, la initializare i se poate specifica cat valoarea cu care sa fie incrementat, deasemenea si o valoare dafault. In cazul in care utilizatorul introduce o valoare eronata, se afiseaza un mesaj de eroare.

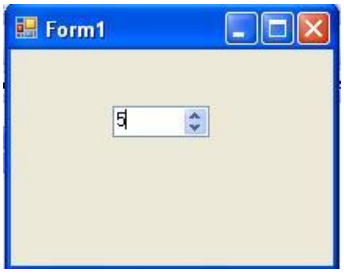


Figura 6 - "Numeric Up Down" in Windows Forms

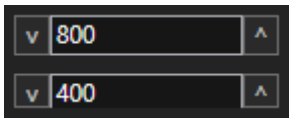


Figura 7 - "Numeric Up Down" customizat

Figura 8 - "Numeric Up Down" customizat cu mesaj de eroare

Ca orice control din WPF, si acest “numeric up down” emite un eveniment atunci cand valoarea s-a schimbat, sau cand unul dintre cele doua butoane a fost apasat cu ajutorul unui “event” bazat pe un “delegate” care trimite la observator instanta controler-ului plus valoarea noua pe care a luat-o. Cele doua functii `private void IncrementUp_Event(object sender, RoutedEventArgs e)` si `private void IncrementDown_Event(object sender, RoutedEventArgs e)` sunt apelate la randul lor de evenimentul „Click” ale celor doua butoane.

```
public delegate void ValueChangedFunction(object sender, double value);
public event ValueChangedFunction ValueChanged;

private void IncrementUp_Event(object sender, RoutedEventArgs e)
{
    CurrentValue += IncrementValue;
    IncrementTextControl.Text = CurrentValue.ToString();
    ValueChanged?.Invoke(this, CurrentValue);
}

private void IncrementDown_Event(object sender, RoutedEventArgs e)
{
    CurrentValue -= IncrementValue;
    IncrementTextControl.Text = CurrentValue.ToString();
    ValueChanged?.Invoke(this, CurrentValue);
}
```

Figure 9 - Cod ce trateaza evenimentul "Click" pe cele doua butoane din controler

“VideoFilesUserControl” este un alt control customizat necesar pentru a stoca informatiile de baza ale fisierelor video incarcate. Este formata dintr-o iconita, numele fisierului video in partea de sus, iar jos informatiile de baza: durata video-ului si numarul de cadre pe secunda. Toate aceste campuri sunt umplute la intializarea controler-ului.

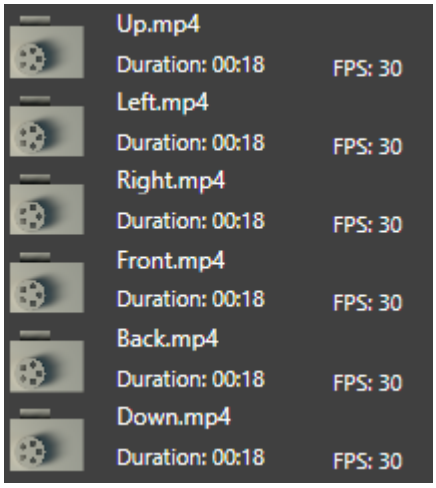


Figura 10 - Cele 6 controale de tip "VideoFilesUserControl"

Dupa ce user-ul isi incarca toate fisierele video, urmatorul cod este rulat unde sunt initializate 6 controale de acest tip pentru fiecare video in parte.

```
for (int i = 0; i < 6; i++)
{
    VideoFileUserControl videoFileUserControl = new VideoFileUserControl(_resouceManager.GetAllVideoPaths()
    videoFileUserControl.VideoClicked += Vfuc_VideoClicked;
    CubeMapListVideos.Children.Add(videoFileUserControl);
```

Figure 11 - Cod ce adauga un nou "VideoFilesUserControl" pe fereastra principala

Pentru a prelua toate aceste informatii, se foloseste libraria “VideoInformationRetrieval” ce returneaza o instanta a clasei “VideoInformation” din libraria “Entities”. Aceasta clasa contine viteza cadrelor pe secunda, numarul total de cadre din video si durata acestuia. In spatele acestui controler avem o functie publica care permite setarea tuturor acelor informatii. Deasemenea, cand utilizatorul da click pe acest controller in interfata, se schimba culoarea de background a acestuia, iar in codul din spatele paginii “EditorPage” exista o logica care permite doar un video controller sa fie selectat.

```
public void SetVideoInformation(string p)
{
    VideoInformation vi = new VideoInformationRetrieval.VideoInformationRetrieval().GetVideoInformation(FilePath);

    durationLabel.Content = duration +
        (vi.duration.Minutes < 10 ? "0" + vi.duration.Minutes : vi.duration.Minutes.ToString())
        + ":" + (vi.duration.Seconds < 10 ? "0"
        + vi.duration.Seconds : vi.duration.Seconds.ToString());
```

Figure 12 - Cod ce initializeaza informatiile necesare unui control de tip "VideoFilesUserControl"

```
namespace Entities
{
    public class VideoInformation
    {
        public int frameRate;
        public long frames;
        public TimeSpan duration;
    }
}
```

Figure 12 - Clasa "VideoInformation" din "assembly"-ul "Entities"

```
public void ResetBackground()
{
    this.Background = new SolidColorBrush(Color.FromArgb(0, 0, 0, 0));
    isSelected = false;
}


private void UserControl_MouseDown(object sender, MouseButtonEventArgs e)
{
    VideoClicked?.Invoke(this);
    this.Background = new SolidColorBrush(Color.FromRgb(25,25,100));
    isSelected = true;
}
```

Figure 13 - Cod ce trateaza schimbarea culorii de fundal al unui "VideoFilesUserControl"

```
private void Vfuc_VideoClicked(object e)
{
    for (int i = 0; i < CubeMapListVideos.Children.Count; i++)
    {
        if (!CubeMapListVideos.Children[i].Equals(e))
        {
            ((VideoFileUserControl)CubeMapListVideos.Children[i]).ResetBackground();
        }
    }
}
```

Figure 13 - Cod ce trateaza evenimentul "Click" atunci cand utilizatorul apasa pe un "VideoFilesUserControl"

Urmatorul control customizat de care va voi vorbi este “VideSeekerUserControl”. Rolul acestuia este de a oferi posibilitatea user-ului de a face “seek” pe video-ul final. Este format dintr-un seeker personalizat (asemanator multor software-uri de editare video) si o cutie in care sunt puse 6 dreptunghiuri rotunjite ce reprezinta durata fiecarui video dat ca input. Daca user-ul apasa click stanga pe acel simbol all seeker-ului, programul va randa proiectia equirectangula la un alt moment *t* din video-ul final.

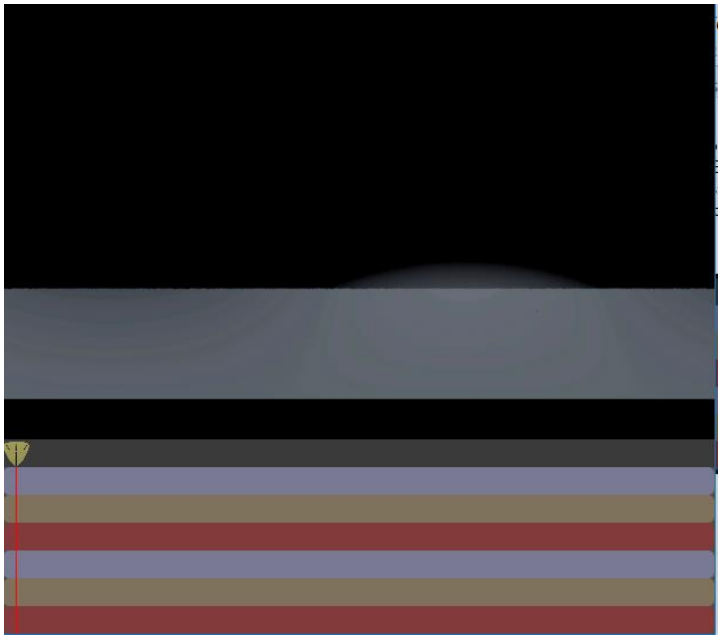


Figure 14 - Secunda la care se afla "seeker"-ul este 0

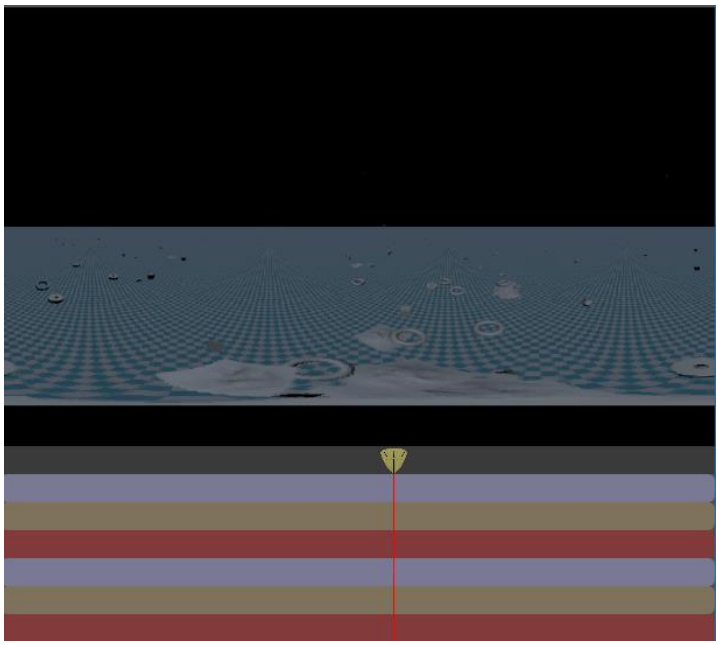


Figure 15 -"Seeker"-ul se afla la mijlocul video-ului final

La baza, acest control are si el un eveniment care anunta obiectele “abonate” ca starea acestuia s-a schimbat. Ca mod de implementare, pozitia acelui seeker este stocata ca o variabila de tip *double* normalizata la intervalul [0, 100]. *Abonatul* trebuie sa se ocupe de intrepretarea acestei valori, in cazul meu, codul din spatele interfetei primeste aceasta valoare, o trasforma intr-o variabila de tip Int reprezentand un anumit numar de secunde, aceasta variabila mergand mai departe la apelul functiei ce randeaza un preview. Evenimentul declarat in controler este asemenator cu cel de la “NumericUpDownController”, trimitand la abonat instanta curenta a controler-ului si noua valoare.

```
public delegate void SeekderChangedFunction(object sender, double value);
public event SeekderChangedFunction SeekerChangedValue;

private void seekerTop_MouseMove(object sender, MouseEventArgs e)
{
    if (e.LeftButton == MouseButtonState.Pressed)
    {
        double mousePoint = e.GetPosition(this.referenceGrid).X;
        if (mousePoint < 0)
        {
            mousePoint = 0 - 5;
        }

        if(mousePoint > this.referenceGrid.ActualWidth)
        {
            mousePoint = referenceGrid.ActualWidth;
        }

        this.seekerGrid.Margin = new Thickness(
            //seekerTop.Margin.Left,
            mousePoint- 5,
            seekerGrid.Margin.Top,
            seekerGrid.Margin.Right,
            seekerGrid.Margin.Bottom
            );
        widthPercentage = (mousePoint * 100.0) / this.referenceGrid.ActualWidth;
    }
}

private void Grid_MouseUp(object sender, MouseButtonEventArgs e)
{
    SeekerChangedValue?.Invoke(this, widthPercentage);
}

private void seekerTop_MouseUp(object sender, MouseButtonEventArgs e)
{
    SeekerChangedValue?.Invoke(this, widthPercentage);
}
```

Figure 16 - Cod ce trateaza schimbarea pozitiei "seeker"-ului

Ultimul user control customizat de care va voi vorbi se numeste “VideoEntryPoint”. Rolul acestuia este de a-i oferi utilizitorului posibilitatea de a sincroniza fisierele video. Pentru fiecare video este incarcat un astfel de controller. La fel, controler-ul dispune de un eveniment care anunta libraria PreviewRenderer ca trebuie sa mai faca un preview in cazul in care utilizatorul a schimbat valoarea acestuia. Ca input, acest controler accepta doar o valoarea de tip TimeSpan. Valoarea default va fi setata la 00:00:00 (hh:mm:ss). Este format dintr-un Label ce va avea ca continut numele fisierului video si un TextBox unde utilizatorul poate sa introduca valoarea dorita.

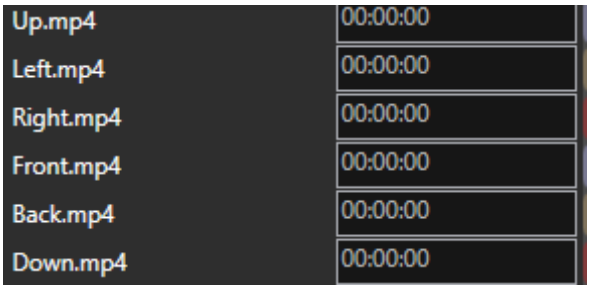


Figure 17 - Controloale de tip "EntryPoint" pentru cele 6 video-uri

```
public partial class VideoEntryPoint : UserControl
{

    public VideoType VideoEntryType { get; set; }

    public delegate void VideoEntryPointChanged(object sender, TimeSpan value);
    public event VideoEntryPointChanged ValueChanged;

    public TimeSpan timeSpan = TimeSpan.FromSeconds(0);

    public VideoEntryPoint(string videoName)
    {
        InitializeComponent();
        this.TimeLapseEntryPoint.Text = TimeSpan.FromSeconds(0).ToString();
        this.VideoName.Content = videoName;
    }

    private void TimeLapseEntryPoint_TextChanged(object sender, TextChangedEventArgs e)
    {
        try
        {
            timeSpan = TimeSpan.Parse(TimeLapseEntryPoint.Text);
            ValueChanged?.Invoke(this, this.timeSpan);
        }
        catch { }
    }
}
```

Figure 18 - Cod ce trateaza schimbarea textului din unul din cele 6 "TextBox"-uri

Implementarea proiectiei Equirectangulare

Implementarea proiectiei equirectangulare am gasit-o pe net intr-o forma accesibila. Modul de transformare a celor 6 imagini este asemanator cu ceea ce am descris la capitolul I ‘Transformarea unei mape cubice intr-o mapa sferica Equirectangulara’.

Codul se foloseste de un bitmap reprezentand mapa cubica, si doua valori reprezentand lungimea si latimea bitmap-ul dat ca output. Mapa cubica o construiesc eu inainte de la apela functia de transformare.

```
private void GenerateCubeMap()
{
    CubeMap = new Bitmap(_singleCubeFaceEdgeDimension * 4, _singleCubeFaceEdgeDimension * 3);
    for (int x = 0; x < CubeMap.Width; x++)
    {
        for (int y = 0; y < CubeMap.Height; y ++)
        {
            if (x >= _singleCubeFaceEdgeDimension * 2 && x < _singleCubeFaceEdgeDimension * 3)
                if (y >= 0 && y < _singleCubeFaceEdgeDimension * 1)
                    CubeMap.SetPixel(x, y, Up.GetPixel(x %( _singleCubeFaceEdgeDimension * 2), y)); //UP

            if (x >= _singleCubeFaceEdgeDimension * 1 && x < _singleCubeFaceEdgeDimension * 2)
                if (y >= _singleCubeFaceEdgeDimension * 1 && y < _singleCubeFaceEdgeDimension * 2)
                    CubeMap.SetPixel(x, y, Left.GetPixel(x % _singleCubeFaceEdgeDimension, y % _singleCubeFaceEdgeDimension * 1));//Left

            if (x >= 0 && x < _singleCubeFaceEdgeDimension * 1)
                if (y >= _singleCubeFaceEdgeDimension * 1 && y < _singleCubeFaceEdgeDimension * 2)
                    CubeMap.SetPixel(x, y, Back.GetPixel(x, y % _singleCubeFaceEdgeDimension * 1));//Back

            if (x >= _singleCubeFaceEdgeDimension * 2 && x < _singleCubeFaceEdgeDimension * 3)
                if (y >= _singleCubeFaceEdgeDimension * 1 && y < _singleCubeFaceEdgeDimension * 2)
                    CubeMap.SetPixel(x, y, Front.GetPixel(x %( _singleCubeFaceEdgeDimension * 2), y % _singleCubeFaceEdgeDimension * 1));//Front

            if (x >= _singleCubeFaceEdgeDimension * 3 && x < _singleCubeFaceEdgeDimension * 4)
                if (y >= _singleCubeFaceEdgeDimension * 1  && y < _singleCubeFaceEdgeDimension * 2)
                    CubeMap.SetPixel(x, y, Right.GetPixel(x % (_singleCubeFaceEdgeDimension * 3), y % _singleCubeFaceEdgeDimension * 1));//Right

            if (x >= _singleCubeFaceEdgeDimension *2 && x < _singleCubeFaceEdgeDimension * 3)
                if (y >= _singleCubeFaceEdgeDimension * 2 && y < _singleCubeFaceEdgeDimension * 3)
                    CubeMap.SetPixel(x, y, Down.GetPixel(x % (_singleCubeFaceEdgeDimension * 2), y % (_singleCubeFaceEdgeDimension * 2)));//Down
        }
    }
}
```

Figure 19 - Cod ce transforma 6 imagini intr-o mapa cubica

Libraria AFORGE

Aforge este o librarie open-source construita special pentru .NET Framework pentru procesare video si inteligenta artificiala. Pentru programul meu, am folosit pachetul Aforge.FFMPEG pentru procesarea fisierelor video.

Pachetul precizat mai sus imi ofera doua clase: `VideoFileReader` si `VideoFileWriter` pe care le-am folosit in cele 3 librarii create de mine: PreviewRenderer, RenderEngine si VideoInformationRetrieval.

Avantaje

Cele doua clase precizate mai sus lucreaza in spate cu obiectul `System.Drawing.Bitmap`. Acest obiect este usor de utilizat in cod, in special datorita celor doua functii SetPixel() si GetPixel().

Pentru deaschiderea unui fisier video este necesara doar calea pe disk a fisierului. Obiectul VideoFileReader imi pune la dispozitie urmatoarele informatii utile: lungime, latime, cadre pe secunda, numarul total de cadre.

Dezavantaje

Desi aceasta librarie este usor de utilizat, are si cateva dezanvataje.

Programul meu ofera posibilitatea de a face ‘seek’ (previzualizarea) unui anumit cadru din video-ul final. Libraria nu suporta aceasta functionalitate, nu imi poate returna direct un cadru precizat de mine, sau o functie de genul “Bitmap GetFrameAtIndex(int index);”. Libraria are o singura functie pentru asa ceva si anume “Bitmap ReadVideoFrame();”. Aceasta functie lucreaza in felul urmator: apeland o singura data functia imi returneaza primul cadru din video, apeland inca o data functia, imi returneaza cadrul din video. Asadar, in cod, pentru a face preview la un anumit cadru *n*, am fost nevoit sa apelez aceasta functie de *n-1* si apoi la apelul *n* sa imi returneze cadrul dorit. Toate aceste apeluri ingreuneaza aplicatia.

Tot din cauza modalitatii de a accesa un anumit cadru din video, am avut probleme cu memoria Heap. Aplicatia mea fiind compilata pe arhitectura de 32 de biti, framework-ul .NET nu permite alocarea unei zone atat de mare de memorie intr-un timp atat de scurt. Toate ce le *n* apeluri ale functiei “ReadVideoFrame()” creeaza in spate *n* bitmap-uri care raman in memorie. “Garbage Cleaner”-ul nu intervine imediat pentru a elibera acea zona de memorie. La un moment dat, constructorul obiectului System.Drawing.Bitmap arunca o exceptie cu textul “Parameter is not valid”. In acest moment CLR refuza sa mai aloce memorie pentru inca o instanta a obiectului Bitmap. Pentru a rezolva aceasta problema, am apelat fortat “Garbage Cleaner”-ul o data la 50 de apeluri ale functiei.

```
try
{
    for (int i = 0; i < index; i++)
    {
        base.ReadVideoFrame();
        if (i % 50 == 0) System.GC.Collect();
    }
}
catch(Exception e)
{

}
```

Figure 20 - Cod ce returneaza un cadru i din video

Libraria nu imi ofera optiuni la salvarea unui fisier video, si anume, nu pot seta tipul codificarii (“encoding”), si nici numarul de cadre pe secunda. Clasa `VideoFileWriter` are o functie numita „WriteVideoFrame” ce primeste ca parametru un cadru (Bitmap) si un obiect de tip TimeSpan ce reprezinta momentul *t* cand acel cadru este afisat. In cod, am avut grija ca fiecare secunda din video-ul final sa contina 30 de cadre (30 fps).

```
TimeSpan fps30 = TimeSpan.FromMilliseconds(33);
```

Figure 21 - Initializarea intervalului pentru un video cu o viteza de 30 de cadre pe secunda

```
FinalVideoOutputWriter.WriteVideoFrame(equirect, TimeSpan.FromMilliseconds(fps30.Milliseconds * i));
```

Figure 22 - Scrierea unui cadru la un interval de 33 de milisecunde

Interfata UI

Fereastra principala

La deschiderea programului, interfata nu imi arata nimic in imaginea de preview. Toate elementele vor fi incarcate atunci cand utilizatorul isi incarca cele 6 video-uri. Interfata se deschide pe tot ecranul initial. Comunicarea intre pagini sau ferestre nu se face atat de atat de usor, nici in Windows Forms si nici in WPF. Pentru a rezolva aceasta problema, am creat o clasa de tip “Singleton”, numita “ResourceManager”. Scopul acesteia este de a stoca informatii privind video-urile introduse, tipul acestora si setarile video-ului final.

Figure 23 - Interfata UI

```
public class ResourceManager
{
    #region Singleton Logic
    static ResourceManager Instance;
    private ResourceManager()
    {

    }

    public static ResourceManager GetInstance()
    {
        if (Instance == null)
        {
            Instance = new ResourceManager();
        }

        return Instance;
    }
    #endregion

    #region Settings
    public int previewRenderWidth { get; set; } = 800;
    public int previewRenderHeight { get; set; } = 400;
    public int outputWidth { get; set; } = 1600;
    public int outputHeight { get; set; } = 800;
    public TimeSpan outputVideoDuration { get; set; } = TimeSpan.FromSeconds(30);
    #endregion;

    private Dictionary<string, VideoType> VideoPaths = new Dictionary<string, VideoType>();

    private Bitmap FrontVideo;
    private Bitmap BackVideo;
    private Bitmap LeftVideo;
    private Bitmap RightVideo;
    private Bitmap UpVideo;
    private Bitmap DownVideo;
```

Figure 24 - Clasa "ResourceManager" cu toate proprietatile

Aplicatia are doar o singura pagina care este folosita, si anume “EditorPage”. “CubeMapEditor1” a fost si ea intial o pagina, dar am transformat-o ulterior intr-o fereastra separata. Toate aceste pagini sunt initializate si afisate de catre “MainWindow”.

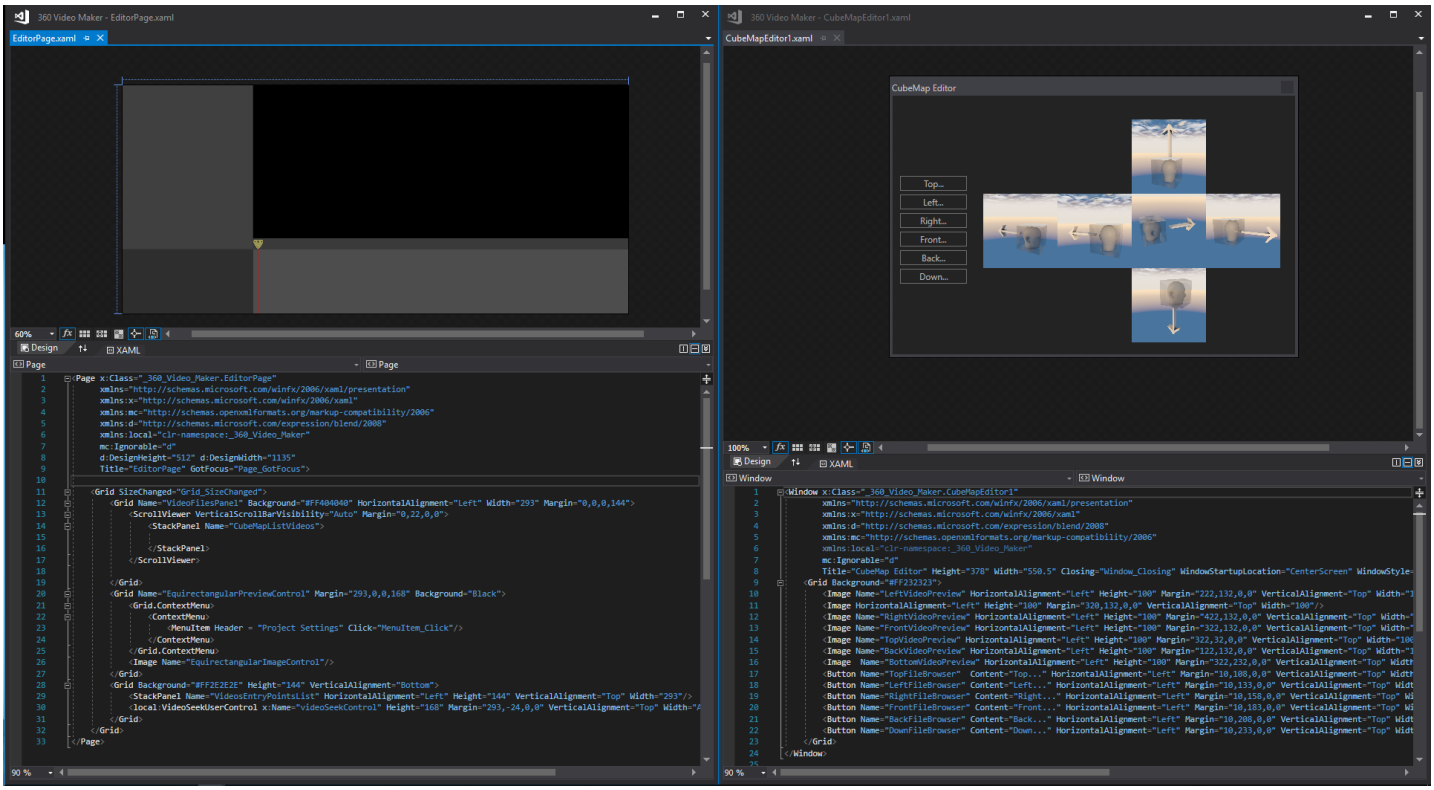


Figure 25 - Fereastra UI si fereastra mapei cubice in editorul XML

```
public partial class MainWindow : Window
{
    private EditorPage EP = new EditorPage();
    private CubeMapEditor1 CME1 = new CubeMapEditor1();

    public MainWindow()
    {
        InitializeComponent();
        WindowStartupLocation = WindowStartupLocation.CenterScreen;
        WindowState = WindowState.Maximized;
        MainFrame.Content = EP;
    }

    private void GoToRenderer_MenuItem(object sender, RoutedEventArgs e)
    {
        //MainFrame.Content = RP;
    }

    private void GoToEditor_MenuItem(object sender, RoutedEventArgs e)
    {
        MainFrame.Content = EP;
    }
```

Figure 26 - Initializarea ferestrei principale si a paginii de editare

In partea de sus, stanga, utilizatorul are la dispozitie un meniu: “Go To” si “Action”. Dand click pe “Go to” utilizatorul poate deschide fereastra numita “Cube Map Editor”.

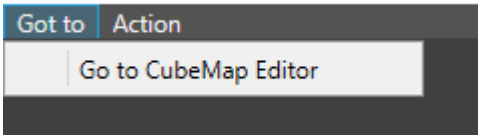


Figure 27 - Meniul pentru "Go to"

```
private void GoToCubeMapEditor_MenuItem(object sender, RoutedEventArgs e)
{
    CME1.ShowDialog(); //cubeMapeEditor
    EP.GotFocus_Custom();
}
```

Figure 28 - Cod ce deschide fereastra mapei cubice

In aceasta fereastra, utilizator isi poate incarca fisierele video. Are la dispozitie 6 butoane, fiecare dintre ele deschizand un “File Browser” unde se cauta fisierul video. In spate, adresa fizica a acelor video-uri este stocata in “ResourceManager” impreuna cu tipul video-ului (“Top, Left, Right, Front, Back, Down”).

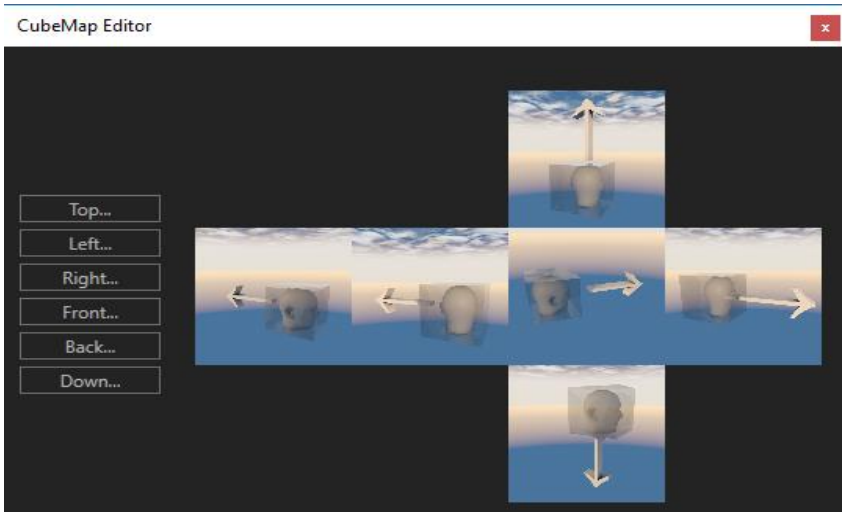


Figure 29 - Fereastra de editare a Mapei Cubice

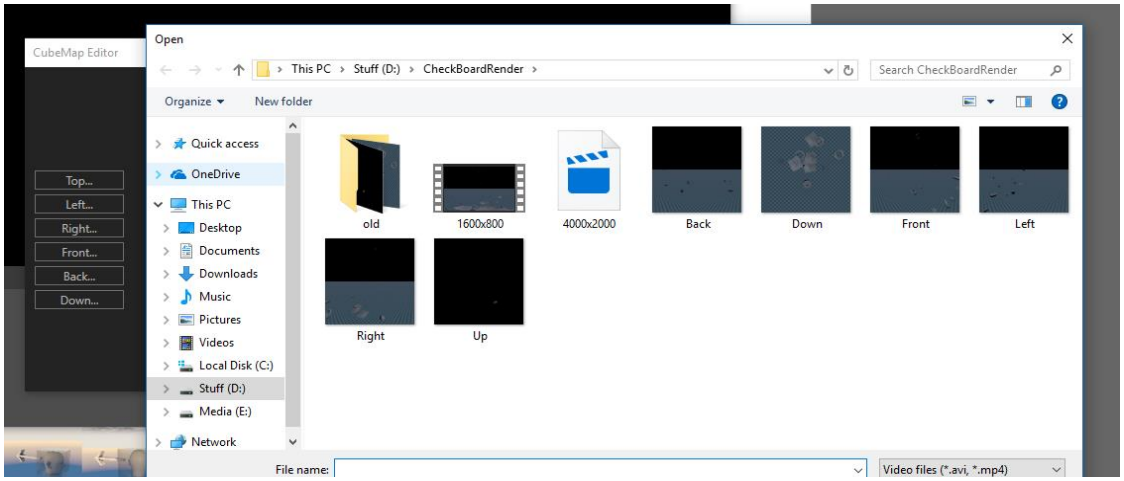


Figure 30 - Deschiderea unui "File Browser" pentru cautarea unui fisier video

```
private void TopFileBrowser_Click(object sender, RoutedEventArgs e)
{
    OpenVideoFile(VideoType.Top);
    TopVideoPreview.Source = GetBitmapImageFrom(_resourceManager.GetPreviewBitmap(VideoType.Top));
}

private void LeftFileBrowser_Click(object sender, RoutedEventArgs e)
{
    OpenVideoFile(VideoType.Left);
    LeftVideoPreview.Source = GetBitmapImageFrom(_resourceManager.GetPreviewBitmap(VideoType.Left));
}

private void RightFileBrowser_Click(object sender, RoutedEventArgs e)
{
    OpenVideoFile(VideoType.Right);
    RightVideoPreview.Source = GetBitmapImageFrom(_resourceManager.GetPreviewBitmap(VideoType.Right));
}

private void FrontFileBrowser_Click(object sender, RoutedEventArgs e)
{
    OpenVideoFile(VideoType.Front);
    FrontVideoPreview.Source = GetBitmapImageFrom(_resourceManager.GetPreviewBitmap(VideoType.Front));
}

private void BackFileBrowser_Click(object sender, RoutedEventArgs e)
{
    OpenVideoFile(VideoType.Back);
    BackVideoPreview.Source = GetBitmapImageFrom(_resourceManager.GetPreviewBitmap(VideoType.Back));
}

private void DownFileBrowser_Click(object sender, RoutedEventArgs e)
{
    OpenVideoFile(VideoType.Bottom);
    BottomVideoPreview.Source = GetBitmapImageFrom(_resourceManager.GetPreviewBitmap(VideoType.Bottom));
}
```

Figure 31 - Cod ce trateaza fiecare buton din fereastra de editare a Mapei Cubice

```
private void OpenVideoFile(VideoType type)
{
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.Filter = "Video files (*.avi, *.mp4) | *.avi; *.mp4";

    if (ofd.ShowDialog() == true)
    {
        _resourceManager.AddVideoFile(ofd.FileName, type);
        _resourceManager.AddVideoEntryPoint(type, TimeSpan.FromSeconds(0));
    }
}
```

Figure 32 - Cod ce trateaza un fisier nou selectat din fereastra "File Browser"

Dupa ficare video adaugat, imaginea din acel cub desfasurat se schimba cu primul cadru din fisierul video. Acele imagini de referinta sunt randate de mine in Cinema 4D; modelul capului nu imi apartine.

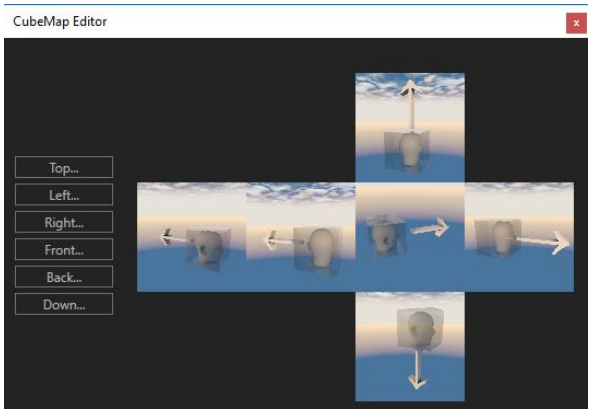


Figure 33 - Mapa Cubica fara nici un video incarcat

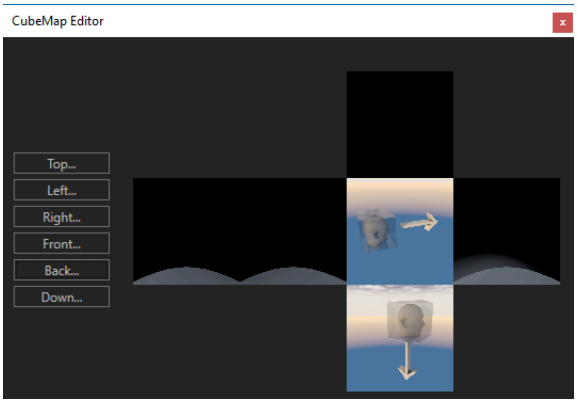


Figure 34 - Mapa cubica cu 4 video-uri incarcate

La inchiderea acestei ferestre, se apeleaza o functie publica din pagina de editare (“GotFocus_Custom()”). Acesta functie semnaleaza faptul ca programul poate incarca elementele visuale si controalele customizate precizate la capitolul 2. Tot aici, se creeaza si prima proiectie a video-ului final.

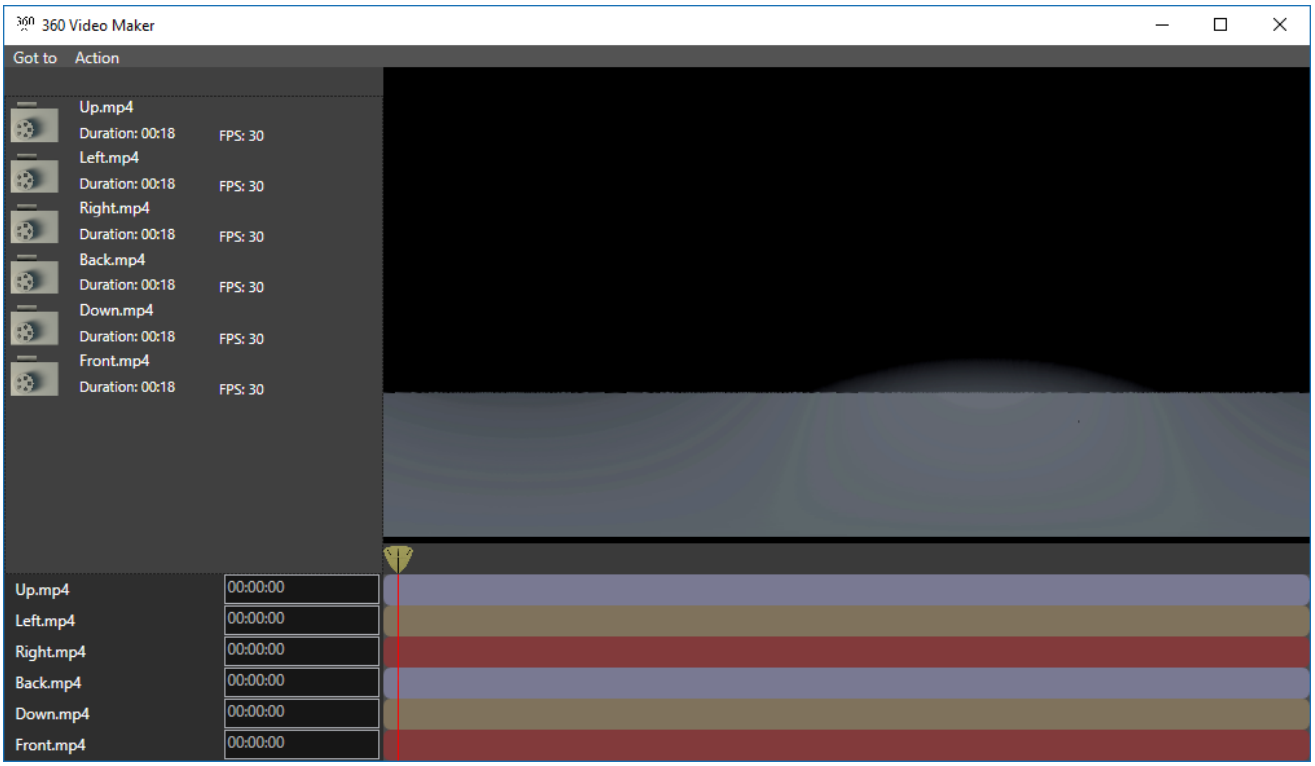


Figure 35 - Fereastra principala dupa adaugarea celor 6 video-uri

```
public void GotFocus_Custom()
{
    Page_GotFocus(null, null);
}

private async void Page_GotFocus(object sender, RoutedEventArgs e)
{
    if (_resouceManager.AllVideosAreLoaded() && videosAllreadyLoaded == false)
    {
        Random r = new Random();
        for (int i = 0; i < 6; i++)
        {
            VideoFileUserControl videoFileUserControl = new VideoFileUserControl(_resouceManager.GetAllVideoPaths().Keys.ElementAt(i));
            videoFileUserControl.VideoClicked += Vfuc_VideoClicked;
            CubeMapListVideos.Children.Add(videoFileUserControl);

            VideoEntryPoint vep = new VideoEntryPoint(System.IO.Path.GetFileName(_resouceManager.GetAllVideoPaths().Keys.ElementAt(i)));
            vep.VideoEntryType = _resouceManager.GetAllVideoPaths().Values.ElementAt(i);
            vep.ValueChanged += Vep_ValueChanged;
            VideosEntryPointsList.Children.Add(vep);

            this.videoSeekControl.AddNewVideoBar(new VideoBar(), colosVideosBar[i % 3]);
        }

        BitmapImage b = GetBitmapImageFrom(await StartPreviewRender());
        EquirectangularImageControl.Source = b;
        videosAllreadyLoaded = true;
        System.GC.Collect();
    }
}
```

Figure 36 - Cod ce adauga pe interfata toate controalele necesare

```
private async Task<Bitmap> StartPreviewRender()
{
    var taskRender = Task<Bitmap>.Run(() =>
    {
        object lockObj = new object(); lock (lockObj)
        {
            return (new PreviewRenderLogic().GetProjection(ResourceManager.GetInstance().previewRenderWidth,
                ResourceManager.GetInstance().previewRenderHeight,
                _seekerSecond));
        }
    });
    return await taskRender;
}
```

Figure 37 - Cod ce returneaza proiectia de la "PreviewRenderer"

Preview Renderer si Render Engine

Preview Renderer

Creearea proiectiei se face asincron (fereastra nu se blocheaza) cu un apel la functia “StartPreviewRender()”. Aceasta la randul ei apeleaza functia “GetProjection” din clasa “PreviewRenderLogic”. Aceasta clasa are scopul de a oferi o fatada a librariei “PreviewRenderer”.

```
public Bitmap GenerateProjection(Dictionary<string, VideoType> videos, int second, Dictionary<VideoType, TimeSpan> videosEntryPoint)
{
    int widthHeight = EquirectangularProjection.Width / 4;

    string frontPath = videos.Single(v => v.Value == VideoType.Front).Key;
    string leftPath = videos.Single(v => v.Value == VideoType.Left).Key;
    string rightPath = videos.Single(v => v.Value == VideoType.Right).Key;
    string upPath = videos.Single(v => v.Value == VideoType.Top).Key;
    string downPath = videos.Single(v => v.Value == VideoType.Bottom).Key;
    string backPath = videos.Single(v => v.Value == VideoType.Back).Key;


    Front = new Bitmap(new VideoInformationRetrieval.VideoInformationRetrieval().
        GetVideoFrame(frontPath, second, videosEntryPoint[VideoType.Front]), _singleCubeFaceEdgeDimension, _singleCubeFaceEdgeDimension);
    Left = new Bitmap(new VideoInformationRetrieval.VideoInformationRetrieval().
        GetVideoFrame(leftPath, second, videosEntryPoint[VideoType.Left]), _singleCubeFaceEdgeDimension, _singleCubeFaceEdgeDimension);
    Right = new Bitmap(new VideoInformationRetrieval.VideoInformationRetrieval().
        GetVideoFrame(rightPath, second, videosEntryPoint[VideoType.Right]), _singleCubeFaceEdgeDimension, _singleCubeFaceEdgeDimension);
    Up = new Bitmap(new VideoInformationRetrieval.VideoInformationRetrieval().
        GetVideoFrame(upPath, second, videosEntryPoint[VideoType.Top]), _singleCubeFaceEdgeDimension, _singleCubeFaceEdgeDimension);
    Down = new Bitmap(new VideoInformationRetrieval.VideoInformationRetrieval().
        GetVideoFrame(downPath, second, videosEntryPoint[VideoType.Bottom]), _singleCubeFaceEdgeDimension, _singleCubeFaceEdgeDimension);
    Back = new Bitmap(new VideoInformationRetrieval.VideoInformationRetrieval().
        GetVideoFrame(backPath, second, videosEntryPoint[VideoType.Back]), _singleCubeFaceEdgeDimension, _singleCubeFaceEdgeDimension);

    GenerateCubeMap();

    return EquirectangularProjection = ConvertToEquirectangular(EquirectangularProjection.Width, EquirectangularProjection.Height);
}
```

Figure 38 - Preluarea a 6 imagini din cele 6 video-uri, crearea mapei cubice si returnarea proiectiei Equirectangulare

Folosind informatiile necesare preluate din “ResourceManager”, fatada apeleaza in continuare “GenerateProjection()” din “PreviewRenderer”.

Diferenta intre cele 6 video-uri o face un obiect enum numit VideoType. In functie de cele 6 fete ale cubului, se incarca 6 obiecte de tip“Bitmap” folosind caile fizice ale video-urile. Apoi se construieste mapa cubica a celor 6 imagini preluate anterior.

Functia “GetVideoFrame” aflata in libraria “VideoInformationRetrieval” este un “wrapper” peste libraria Aforge.FFMPEG. Toate aceste apeluri ajungi la functia “GetFrameAtIndex()”. Aici, in functie de “entry point”-ul video-ului, se decide daca se va returna o imagine neagra sau se merge mai departe la citirea cadrului din fisier.

```
public Bitmap GetVideoFrame(string filePath, int second, TimeSpan timeSpan)
{
    VideoFileReader VFR = new VideoFileReader();
    try
    {
        VFR.Open(filePath);
    }
    catch(Exception e)
    {
        VFR.Close();
        VFR.Dispose();
    }

    var ret = VFR.GetFrameAtIndex(second, timeSpan);
    VFR.Close();
    VFR.Dispose();
    System.GC.Collect();
    return ret;
}
```

Figure 39 - Cod apelat in "VideoInformationRetrieval" pentru returnarea unui cadru

```
public Bitmap GetFrameAtIndex(int second, TimeSpan ts)
{
    int index = 0;
    TimeSpan videoDuration  = TimeSpan.FromSeconds((double)this.FrameCount / this.FrameRate);

    if (ts.Add(TimeSpan.FromSeconds(second)).Seconds > videoDuration.Seconds)
    {
        return CreateBlackBitmap(this.Width, this.Height);
    }

    if (ts.Add(TimeSpan.FromSeconds(second)).Seconds < 0)
    {
        return CreateBlackBitmap(this.Width, this.Height);
    }

    index = ts.Add(TimeSpan.FromSeconds(second)).Seconds * this.FrameRate;

    try
    {
        for (int i = 0; i < index; i++)
        {
            base.ReadVideoFrame();
            if (i % 50 == 0) System.GC.Collect();
        }
    }
    catch(Exception e)
    {

    }

    var ret = ReadVideoFrame();

    this.Dispose();

    return ret;
}
```

Figure 40 - Cod ce ia un cadru la o anumita secunda dintr-un fisier video

```
    return EquirectangularProjection = ConvertToEquirectangular(EquirectangularProjection.Width, EquirectangularProjection.Height);
}

private void GenerateCubeMap()...

/// https://stackoverflow.com/questions/34250742/converting-a-cubemap-into-equirectangular-panorama
public Bitmap ConvertToEquirectangular(int outputWidth, int outputHeight)...
```

Figure 42 - Apelul functiei de transformare a proiectiei Equirectangulare

```
public Bitmap ConvertToEquirectangular(int outputWidth, int outputHeight)
{
    Bitmap equiTexture = new Bitmap(outputWidth, outputHeight);
    double u, v; //Normalised texture coordinates, from 0 to 1, starting at lower left corner
    double phi, theta; //Polar coordinates
    int cubeFaceWidth, cubeFaceHeight;

    cubeFaceWidth = CubeMap.Width / 4; //4 horizontal faces
    cubeFaceHeight = CubeMap.Height / 3; //3 vertical faces


    for (int j = 0; j < equiTexture.Height; j++)
    {
        //Rows start from the bottom
        v = 1 - ((double)j / equiTexture.Height);
        theta = v * Math.PI;

        for (int i = 0; i < equiTexture.Width; i++)
        {
            //Columns start from the left
            u = ((double)i / equiTexture.Width);
            phi = u * 2 * Math.PI;

            double x, y, z; //Unit vector
            x = Math.Sin(phi) * Math.Sin(theta) * -1;
            y = Math.Cos(theta);
            z = Math.Cos(phi) * Math.Sin(theta) * -1;

            double xa, ya, za;
            double a;
            double mx, my, mz;
            //a = Math.Max(new double[3] { Math.Abs(x), Math.Abs(y), Math.Abs(z) });
            mx = Math.Abs(x);
            my = Math.Abs(y);
            mz = Math.Abs(z);

            if (mx > my && mx > mz)
                a = mx;
            else
                if (my > mx && my > mz)
                a = my;
            else
                a = mz;

            //Vector Parallel to the unit vector that lies on one of the cube faces
            xa = x / a;
            ya = y / a;
            za = z / a;

            Color color;
            int xPixel, yPixel;
            int xOffset, yOffset;

            if (xa == 1)
            {
                //Right
                xPixel = (int)((((za + 1f) / 2f) - 1f) * cubeFaceWidth);
                xOffset = 3 * cubeFaceWidth; //Offset
                yPixel = (int)((((ya + 1f) / 2f)) * cubeFaceHeight);
                yOffset = cubeFaceHeight; //Offset
```

Figure 41 - Codul ce returneaza o proiectie Equirectangulara (partea 1)

```
            }
            else if (xa == -1)
            {
                //Left
                xPixel = (int)((((za + 1f) / 2f)) * cubeFaceWidth);
                xOffset = cubeFaceWidth;
                yPixel = (int)((((ya + 1f) / 2f)) * cubeFaceHeight);
                yOffset = cubeFaceHeight;
            }
            else if (ya == 1)
            {
                //Up
                xPixel = (int)((((xa + 1f) / 2f)) * cubeFaceWidth);
                xOffset = 2* cubeFaceWidth;
                yPixel = (int)((((za + 1f) / 2f) - 1f) * cubeFaceHeight);
                yOffset = 2 * cubeFaceHeight;
            }
            else if (ya == -1)
            {
                //Down
                xPixel = (int)((((xa + 1f) / 2f)) * cubeFaceWidth);
                xOffset = 2* cubeFaceWidth;
                yPixel = (int)((((za + 1f) / 2f)) * cubeFaceHeight);
                yOffset = 0;
            }
            else if (za == 1)
            {
                //Front
                xPixel = (int)((((xa + 1f) / 2f)) * cubeFaceWidth);
                xOffset = 2* cubeFaceWidth;
                yPixel = (int)((((ya + 1f) / 2f)) * cubeFaceHeight);
                yOffset = cubeFaceHeight;
            }
            else if (za == -1)
            {
                //Back
                xPixel = (int)((((xa + 1f) / 2f) - 1f) * cubeFaceWidth);
                xOffset = 0;
                yPixel = (int)((((ya + 1f) / 2f)) * cubeFaceHeight);
                yOffset = cubeFaceHeight;
            }
            else
            {
                //Debug.LogWarning("Unknown face, something went wrong");
                xPixel = 0;
                yPixel = 0;
                xOffset = 0;
                yOffset = 0;
            }
            xPixel = Math.Abs(xPixel);
            yPixel = Math.Abs(yPixel);
            xPixel += xOffset;
            yPixel += yOffset;
            if (yPixel >= outputHeight)
                yPixel -= 1;
            if (xPixel >= outputWidth)
                xPixel -= 1;
            color = CubeMap.GetPixel(xPixel, yPixel);
            equiTexture.SetPixel(i, j, color);
        }
    }
    return equiTexture;
```

Figure 43 - Codul ce returneaza o proiectie Equirectangulara (partea 2)

Dupa terminarea proiectiei, executia codului revine in “EditorPage” unde “Bitmap”-ul proiectiei este incarcat in controlul de tip “Image” definit in xml-ul Interfetei. Problema care intervine este urmatorea: contrulul “Image” din WPF accepta ca imagine sursa un obiect “BitmapImage”, insa proiectia vine sub forma unui obiect “Bitmap”. Cele doua obiecte sunt diferite. Nu exista o conversie directa sau un “cast”-ing de la un obiect la altul. Pentru a rezolva aceasta problema am preluat de pe internet o functie care face aceasta conversie din “Bitmap” in “BitmapImage” (“GetBitmapImageFrom(Bitmap b)”).

```
BitmapImage b = GetBitmapImageFrom(await StartPreviewRender());
EquirectangularImageControl.Source = b;
```

Figure 44 - Actializarea controlului de tip "Image" cu noua proiectie generata

```
/// https://stackoverflow.com/questions/6484357/converting-bitmapimage-to-bitmap-and-vice-versa
private BitmapImage GetBitmapImageFrom(Bitmap b)
{
    BitmapImage bitmapImage = new BitmapImage();
    using (MemoryStream memory = new MemoryStream())
    {
        b.Save(memory, ImageFormat.Png);
        memory.Position = 0;

        bitmapImage.BeginInit();
        bitmapImage.StreamSource = memory;
        bitmapImage.CacheOption = BitmapCacheOption.OnLoad;
        bitmapImage.EndInit();
    }
    return bitmapImage;
}
```

Figure 45 - Cod ce transforma un obiect de tip "Bitmap" intr-un obiect de tip "BitmapImage"

Fereastra de setari

Utilizatorul poate seta rezolutia proiectiei, in functie de performantele masinii pe care o detine. Dand click dreapta pe proiectie, se deschide un meniu contextual, unde utilizatorul are acces la diverse setari.

Figure 46 - Meniul contextual

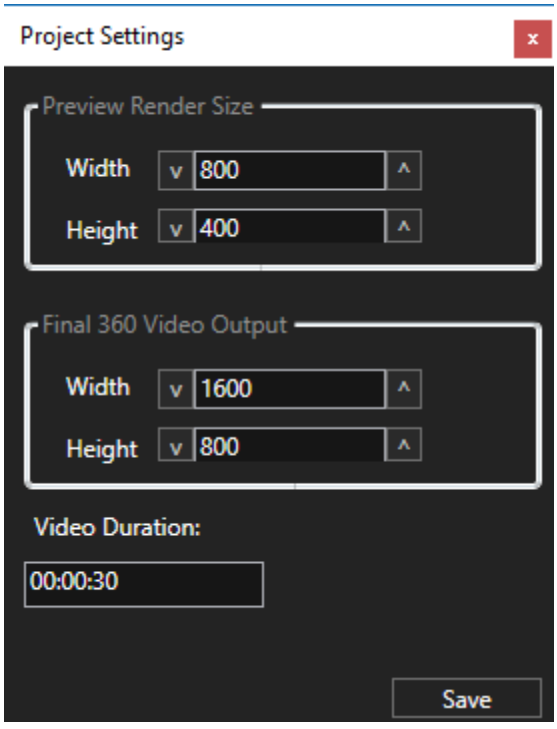


Figure 48 - Fereastra de Setari al programului

```
private void MenuItem_Click(object sender, RoutedEventArgs e)
{
    projSettingsWindow.ShowDialog();
}
```

Figure 47 - Cod ce deschide fereastra de Setari

In aceasta fereastra, utilizatorul poate seta rezolutia proiectiei de preview. Aceasta rezolutie are un impact asupra performantei: cu cat rezolutia este mai mica, cu atat se randeaza mai repede. Tot aici se poate seta rezolutia video-ului final. Pentru o calitate cat mai mare a video-ului final, trebuie sa analizam un pic video-urile date ca input. Raportul dintre lungime si latime trebuie sa fie de 2:1. Lungimea video-ului final ar trebui sa aiba valoarea lungimii unui video dat ca input, inmultit cu 4. Latimea in schimb trebuie sa fie de 2 ori mai mare decat lungimea unui video dat ca input. Ca mod de testare al programului, am randat in Cinema 4D 6 video-uri, fiecare avand o rezolutie de 1000x1000 pixeli. In acest caz, video-ul final ar trebui sa aiba o rezolutie de 4000x2000 pixeli.

Ca si observatie, cu cat rezolutia video-ul final este mai mare, timpul de randare creste.

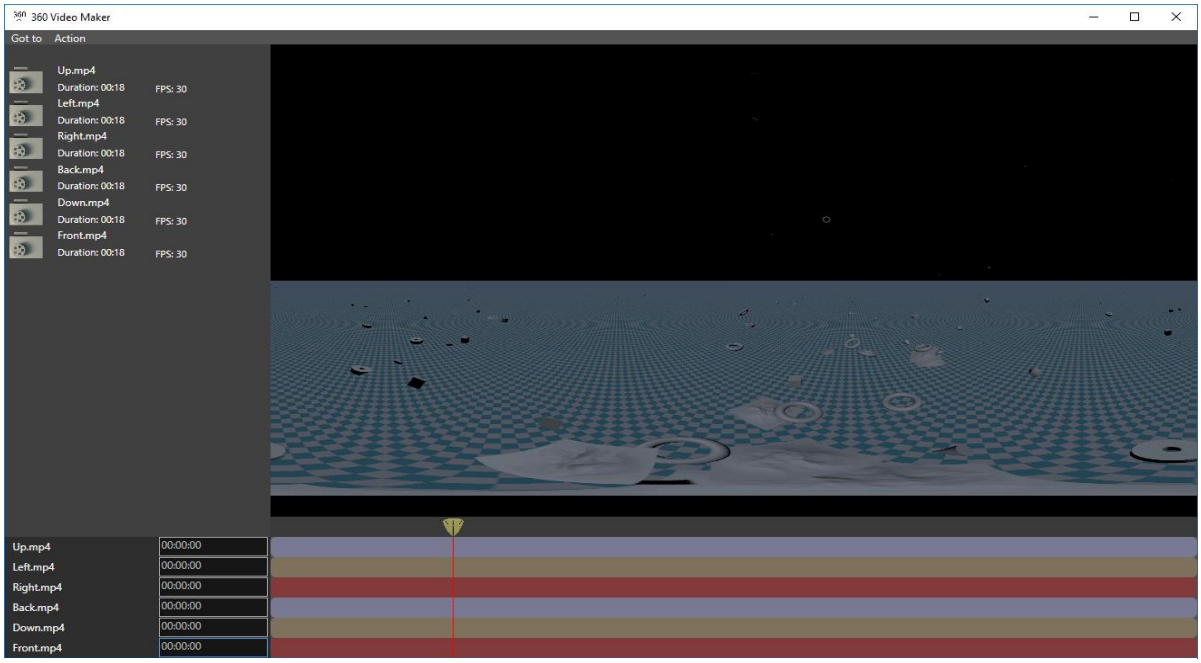


Figure 49 - Proiectie la rezolutia: 4000 x 2000

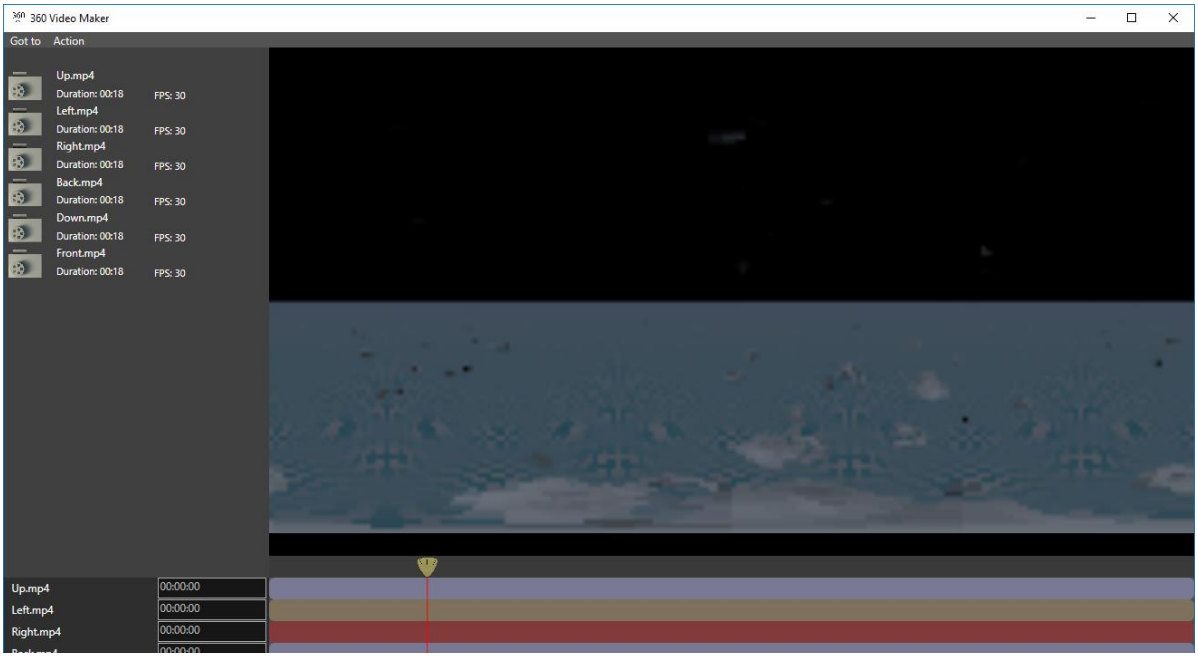


Figure 50 - Proiectie la rezolutia: 200 x 100

Ultimul camp pe care user-ul il poate seta este durata video-ului final. Toate aceste informatii se salveaza in “ResourceManager” in momentul in care apas butonul de “Save”; tot atunci si fereastra se inchide.

```
public partial class ProjectSettings : Window
{
    public ProjectSettings()
    {
        InitializeComponent();

        previewWidth.SetCurrentValue(800);
        previewHeight.SetCurrentValue(400);
        previewWidth.IncrementValue = 1;
        previewHeight.IncrementValue = 1;

        outputHeight.IncrementValue = 1;
        outputWidth.IncrementValue = 1;
        outputHeight.SetCurrentValue( 800);
        outputWidth.SetCurrentValue( 1600);

        outputVideoDuration.Text = TimeSpan.FromSeconds(30).ToString();
    }

    private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
    {
        e.Cancel = true;
        this.Hide();
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        ResourceManager.GetInstance().previewRenderHeight = (int)previewHeight.CurrentValue;
        ResourceManager.GetInstance().previewRenderWidth = (int)previewWidth.CurrentValue;
        ResourceManager.GetInstance().outputHeight = (int)outputHeight.CurrentValue;
        ResourceManager.GetInstance().outputWidth = (int)outputWidth.CurrentValue;
        ResourceManager.GetInstance().outputVideoDuration = TimeSpan.Parse(outputVideoDuration.Text);

        this.Hide();
    }
}
```

Figure 51 - Proprietati si functii principale din spatele ferestrei de Setari

Sincronizarea video-urilor

Trecem mai departe la sincronizarea video-urilor in cazul in care avem nevoie sa facem asta. In partea din stanga jos a ferestrei, langa “VideoSeeker”-ul prezentat la capitolul, utilizatorul are la dispozitie 6 campuri pentru cele 6 video-uri. In aceste campuri, utilizatorul poate seta “entry point”-ul fiecarui video. Valoare pe care o poate introduce este de tip “TimeSpan” si poate fi pozitiva (00:00:05) sau negativa (-00:00:03). O valoarea pozitiva inseamna ca la momentul $t = 0$, cadrul returnat de video se afla cu $t + s$ secunde in fata, unde s este numarul se secunde/minute/ore specificat de utilizator. O valoarea negativa este exact contrariul, la momentul $t = 0$, portiunea din proiectie pentru acel video va fi neagra, deoarece video-ul intra s secunde/minute/ore mai tarziu. Daca durata video-ului final este mai mare decat durata video-urilor date ca input, in ultima parte din video nu se va vedea nimic, va fi negru. Toata aceasta logica exista atat in “PreviewRenderer” cat si in “RenderEngine”.

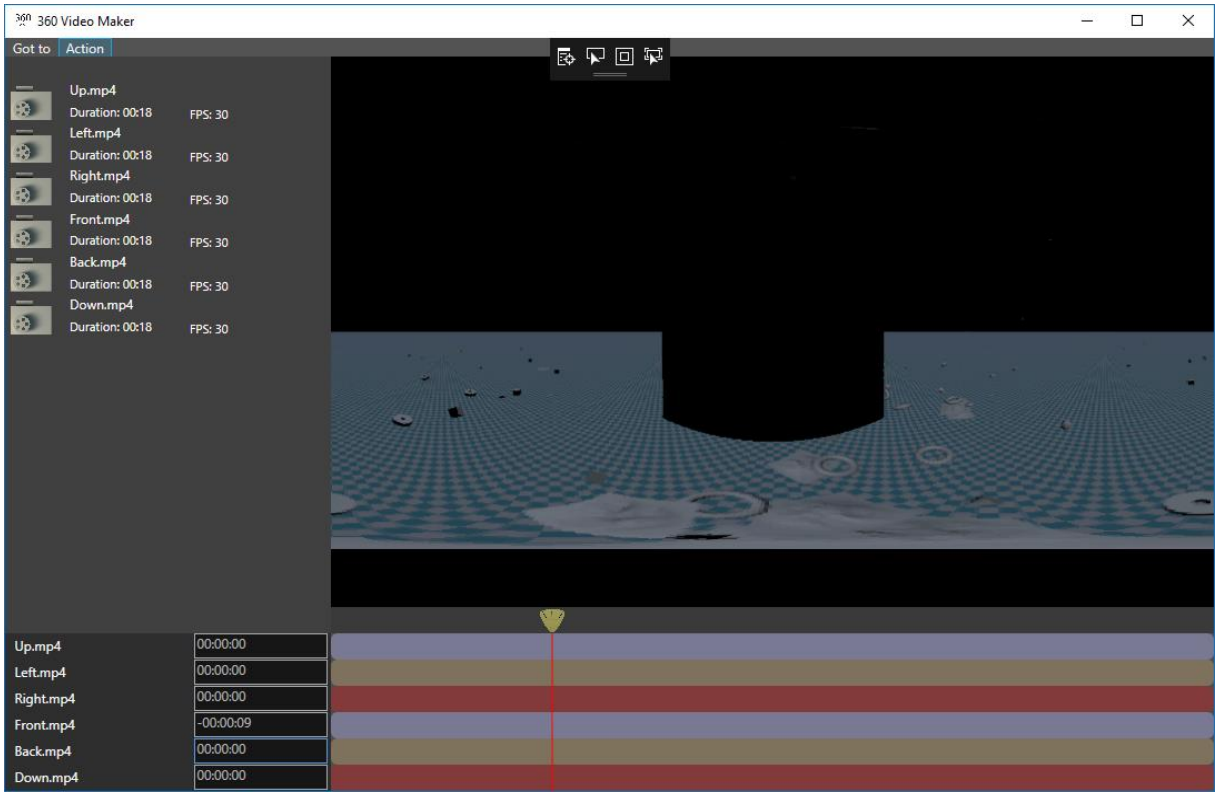


Figure 52 - Video-ul de tip "Front" (fata) este redat intarziat cu 9 secunde fata de celelalte

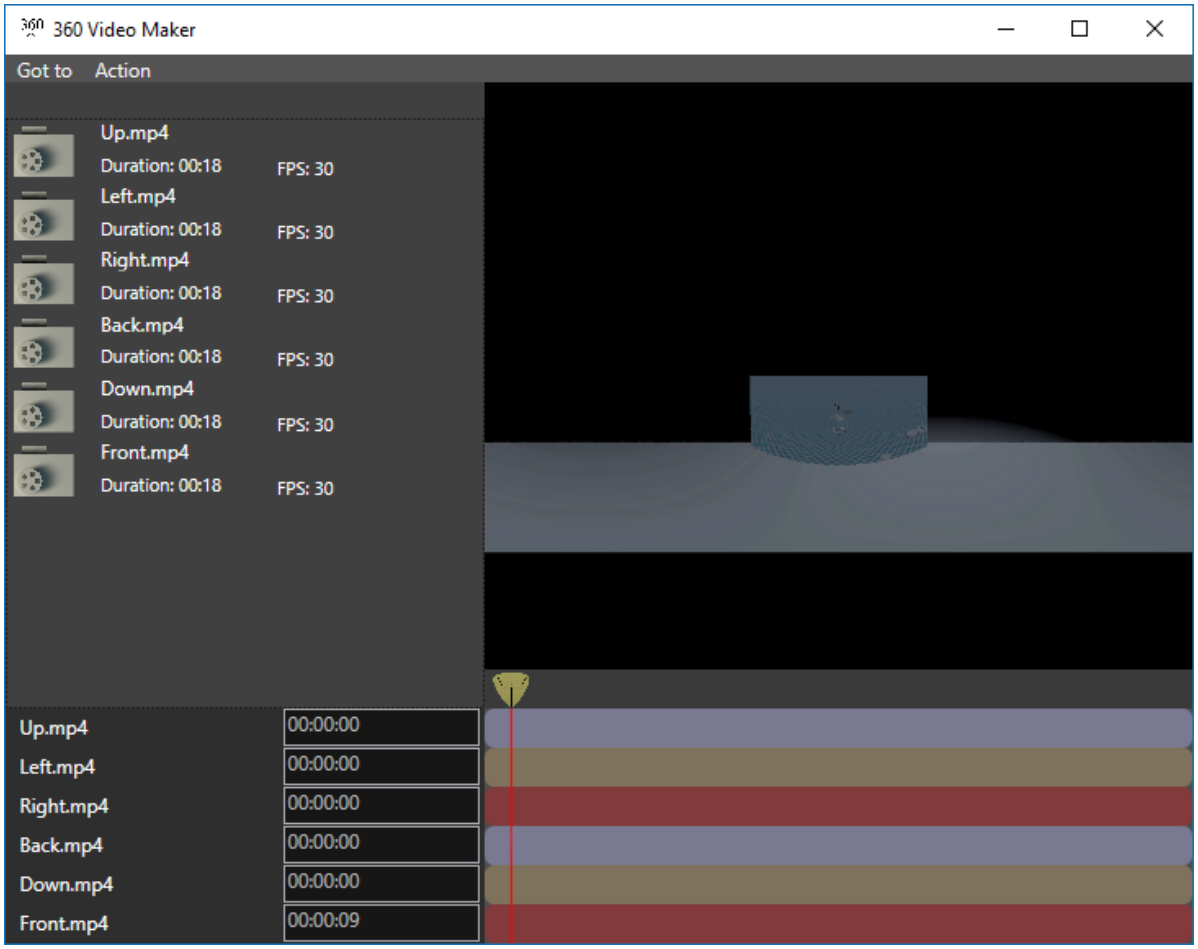


Figure 53 - Video-ul de tip "Front" (fata) este redat in avans cu 9 secunde fata de celelalte

La orice schimbare a unui anumit “entry point”, se creeaza o noua proiectie.

```
private async void Vep_ValueChanged(object sender, TimeSpan value)
{
    _resouceManager.AddVideoEntryPoint(((VideoEntryPoint)sender).VideoEntryType, value);
    BitmapImage b = GetBitmapImageFrom(await StartPreviewRender());
    EquirectangularImageControl.Source = b;
}
```

Figure 54 - Tratarea evenimentului cand un "EntryPoint" isi schimba valoarea

Render Engine

Dupa ce utilizatorul a terminat de editat, poate da drumul la randare. Acest lucru se face tot din meniu dand click pe “Action” apoi “Render”.

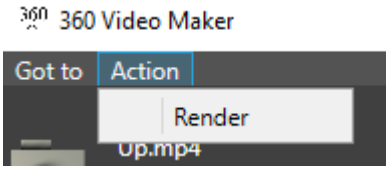


Figure 55 - Meniul Action cu submeniul "Render"

```
private async void MenuItem_Click(object sender, RoutedEventArgs e)
{
    //render click
    var t = Task.Run(() => {
    RenderEngineFacade renderEngineFacade = new RenderEngineFacade();
    renderEngineFacade.Intiliaze();
    renderEngineFacade.Start();
    });
    await t;
}
```

Figure 56 - Taskul asincron pornit din fereastra principala

Procesul de randare se face asincron, in acest fel, utilizatorul poate folosi programul in continuare. In cod se creeaza o noua instanta a fatadei, apoi se initializeaza datele necesare. Din dorinta de a reduce coeziunea intre UI si bussiness logic, am trasmis toate informatiile necesare din “ResourceManager” in “RenderEngine” folosind o clasa “RenderEngineTransferData” din libraria “Entities”.

```
public class RenderEngineTransferData
{
    public Dictionary<string, VideoType> Videos { get; set; }
    public int OutputWidth { get; set; }
    public int OutputHeight { get; set; }
    public TimeSpan VideoDuration { get; set; }
    public Dictionary<VideoType, TimeSpan> VideosEntryPoints { get; set; }

    public string OutputFilePath { get; set; }
}
```

Figure 57 - Clasa "RenderEngineTransferData" cu proprietatile folosite

In functia “Initialize()” se intileaza o instanta a acestei clase, campurile fiind preluate din “ResourceManager”.

```
private RenderEngineTransferData usefullData = new RenderEngineTransferData();

public void Intiliaze()
{
    SaveFileDialog saveFileDialog = new SaveFileDialog();
    saveFileDialog.Filter = "Video files (*.avi, *.mp4) | *.avi; *.mp4";
    if (saveFileDialog.ShowDialog() == true)
    {
        usefullData.OutputFilePath = saveFileDialog.FileName;
    }

    usefullData.OutputHeight = ResourceManager.GetInstance().outputHeight;
    usefullData.OutputWidth = ResourceManager.GetInstance().outputWidth;
    usefullData.VideoDuration = ResourceManager.GetInstance().outputVideoDuration;
    usefullData.Videos = ResourceManager.GetInstance().GetAllVideoPaths();
    usefullData.VideosEntryPoints = ResourceManager.GetInstance().GetVideosEntryPoints();
}
```

Figure 58 - Cod ce initializeaza o instanta a clase "RenderEngineTransferData"

Urmatoarea functie apelata este “Start()” care initializeaza o instanta a librariei “RenderEngine” folosind ca parametru la constructor clasa initializata mai sus, urmat de apelul functiei “StartProcessing()”.

```
public async Task Start()
{
    RenderEngine.RenderEngine re = new RenderEngine.RenderEngine(usefullData);
    await re.StartProcessing();
}
```

Figure 59 - Functia Start din fatada “RenderEngineFacade”

Transformarea dintr-o mapa cubica in una “Equirectangulara” este un proces intens pentru CPU; motivul fiind transformarea coordonatelor polare in coordonate cartieziene. Deoarece toate aceste trasformari se fac la fiecare cadru al video-ului final, am decis sa fac o mapare intre cele doua tipuri de coordonate, folosind o matrice de corespondenta. Astfel, procesorul nu mai face toate acele calcule, transformarea trasformandu-se intr-o serie de citiri si scrieri pe memoria RAM.

Pentru matricea de corespondenta am creat 2 clase interne: “Point” si “CorespondencePoint”. Am preluat codul care face proiectia “Equirectangulara” si l-am modificat putin pentru a stoca maparea dintre un pixel de pe mapa cubica pe mapa “Equirectangulara”. Dupa initializarea acestei matrici, are loc faza de inatarziere a video-ului, in cazul in care utilizatorul a pus valori pozitive in “Entry point”-uri.

```
internal class CorespondencePoint
{
    public Point p1 { get; set; }
    public Point p2 { get; set; }
}

internal class Point
{
    public int x { get; set; }
    public int y { get; set; }
}
```

Figure 60 - Clasa "Point" si "CorespondencePoint"

```
//color = CubeMap.GetPixel(xPixel, yPixel);

//equiTexture.SetPixel(i, j, color);
Equirectangular2CubeMap_Correspondence[j].Add(new CorespondencePoint()
{
    p1 = new Point() { x = i, y = j },
    p2 = new Point() { x = xPixel, y = yPixel }
});
```

Figure 61 - Bucata de cod inlocuita in functia de proiectie "Equirectangulara"

Initial, functia de procesare am vrut sa o fac pe mai multe fire de executie, folosindu-ma de obiectul “Task”. Ca mod de lucru, am creat o lista de “Task”-uri in functie de numarul de fire de executie ale procesorului, apoi pentru fiecare task am creat o mapa cubica, urmand ca task-ul sa creeze mapa “Equirectangulara”. Toate “Task”-urile din lista sunt asteptate la final, si tot atunci, cadrele sunt scrise pe disc. Aceata metoda de procesare nu a reusit sa functioneze asa cum ma asteptam, aplicatia utilizand doar un singur fir de executie. Astfel, am renuntat la acel cod pe mai multe fire de execute si l-am facut doar pe un singur fir.

```
for (int i = 0; i < UsefullData.VideoDuration.Seconds * frameRate; i++)
{
    System.GC.Collect();
   // List<Task<Bitmap>> tasks = new List<Task<Bitmap>>();
    //List<Bitmap> cubemapsForEachThread = new List<Bitmap>();
    //for (int thr = 0; thr < Environment.ProcessorCount; thr++)
    //{
```

Figure 62 - Initializarea listei pentru "Task"-uri

```
//System.GC.Collect();
//  return equirect;
//});
//tasks.Add(task);
//await task;
// }
not threaded
// Task.WaitAll(tasks.ToArray());
// int thr1 = 0;
// foreach (var t in tasks)
//{
//FinalVideoOutputWriter.Open(UsefullData.OutputFilePath, UsefullData.OutputWidth, UsefullData.OutputHeight);
```

Figure 63 - Asteptarea "Task"-urilor si scrierea rezultatelor pe disk

Dupa cum putem obseva, am apelat si aici in mod fortat “Garbage Cleaner”-ul. Singurul caz in care poate sa apara aceasta necesitate ar fi atunci cand rezolutia video-ului de output este mica, iar “Bitmap”-urile se instantiaza foarte repede.

Codul final ce ruleaza pe un singur fir de executie este prezentat in imaginea urmatoare. Pentru fiecare cadru din video-ul final, construiesc cele 6 “Bitmap”-uri, apoi pe baza matricei de corespondenta setez fiecare pixel in parte din proiectia finala. Mapa cubica se creeaza in acelasi stil ca si in “PreviewRenderer”.

```
private async Task RenderThread()
{
    int frameRate = TopFileReader.FrameRate;
    TimeSpan fps30 = TimeSpan.FromMilliseconds(33);
    for (int i = 0; i < UsefullData.VideoDuration.Seconds * frameRate; i++)
    {
        System.GC.Collect();

        Bitmap cubemap = new Bitmap(UsefullData.OutputWidth, (UsefullData.OutputWidth / 4) * 3);
        Bitmap equirect = new Bitmap(UsefullData.OutputWidth, UsefullData.OutputHeight);
        int _singleCubeFaceEdgeDimension = UsefullData.OutputWidth / 4;

        Bitmap top = i + UsefullData.VideosEntryPoints[VideoType.Top].Seconds * frameRate < 0
            || i > TopFileReader.FrameCount ? new Bitmap(TopFileReader.Width, TopFileReader.Height) : TopFileReader.ReadVideoFrame();
        Bitmap down = i + UsefullData.VideosEntryPoints[VideoType.Bottom].Seconds * frameRate < 0
            || i > BottomFileReader.FrameCount ? new Bitmap(BottomFileReader.Width, BottomFileReader.Height) : BottomFileReader.ReadVideoFrame();
        Bitmap front = i + UsefullData.VideosEntryPoints[VideoType.Front].Seconds * frameRate < 0
            || i > FrontFileReader.FrameCount ? new Bitmap(FrontFileReader.Width, FrontFileReader.Height) : FrontFileReader.ReadVideoFrame();
        Bitmap back = i + UsefullData.VideosEntryPoints[VideoType.Back].Seconds * frameRate < 0
            || i > BackFileReader.FrameCount ? new Bitmap(BackFileReader.Width, BackFileReader.Height) : BackFileReader.ReadVideoFrame();
        Bitmap left = i + UsefullData.VideosEntryPoints[VideoType.Left].Seconds * frameRate < 0
            || i > LeftFileReader.FrameCount ? new Bitmap(LeftFileReader.Width, LeftFileReader.Height) : LeftFileReader.ReadVideoFrame();
        Bitmap right = i + UsefullData.VideosEntryPoints[VideoType.Right].Seconds * frameRate < 0
            || i > RightFileReader.FrameCount ? new Bitmap(RightFileReader.Width, RightFileReader.Height) : RightFileReader.ReadVideoFrame();

        CreateCubeMap(cubemap, _singleCubeFaceEdgeDimension, top, down, front, back, left, right);

        for (int j = 0; j < UsefullData.OutputHeight; j++)
        {
            for (int ji = 0; ji < UsefullData.OutputWidth; ji++)
            {
                equirect.SetPixel(
                            Equirectangular2CubeMap_Corespondence[j][ji].p1.x, Equirectangular2CubeMap_Corespondence[j][ji].p1.y,
                            cubemap.GetPixel(Equirectangular2CubeMap_Corespondence[j][ji].p2.x, Equirectangular2CubeMap_Corespondence[j][ji].p2.y
                            ));
            }
        }
        lock (lockObject)
        {
            FinalVideoOutputWriter.WriteVideoFrame(equirect, TimeSpan.FromMilliseconds(fps30.Milliseconds * i));
        }

        System.GC.Collect();
    }
    FinalVideoOutputWriter.Close();
    codcomentat
}
```

Figure 64 - Preluarea a 6 imagini din cele 6 video-uri, crearea mapei cubice si generarea proiectiei Equirectangulare

```
private void CreateCubeMap(Bitmap cubemap, int _singleCubeFaceEdgeDimension, Bitmap top, Bitmap down, Bitmap front, Bitmap back, Bitmap left, Bitmap right)
{
    if (top == null)
    {
        cubemap = new Bitmap(UsefullData.OutputWidth, (UsefullData.OutputWidth / 4) * 3);
        return;
    }
    top = new Bitmap(top, _singleCubeFaceEdgeDimension, _singleCubeFaceEdgeDimension);
    down = new Bitmap(down, _singleCubeFaceEdgeDimension, _singleCubeFaceEdgeDimension);
    front = new Bitmap(front, _singleCubeFaceEdgeDimension, _singleCubeFaceEdgeDimension);
   back  = new Bitmap(back, _singleCubeFaceEdgeDimension, _singleCubeFaceEdgeDimension);
    left = new Bitmap(left, _singleCubeFaceEdgeDimension, _singleCubeFaceEdgeDimension);
    right = new Bitmap(right, _singleCubeFaceEdgeDimension, _singleCubeFaceEdgeDimension);

    for (int x = 0; x < cubemap.Width; x++)
    {
        for (int y = 0; y < cubemap.Height; y++)
        {
            if (x >= _singleCubeFaceEdgeDimension * 2 && x < _singleCubeFaceEdgeDimension * 3)
                if (y >= 0 && y < _singleCubeFaceEdgeDimension * 1)
                    cubemap.SetPixel(x, y, top.GetPixel(x % (_singleCubeFaceEdgeDimension * 2), y)); //UP

            if (x >= _singleCubeFaceEdgeDimension * 1 && x < _singleCubeFaceEdgeDimension * 2)
                if (y >= _singleCubeFaceEdgeDimension * 1 && y < _singleCubeFaceEdgeDimension * 2)
                    cubemap.SetPixel(x, y, left.GetPixel(x % _singleCubeFaceEdgeDimension, y % _singleCubeFaceEdgeDimension * 1));//Left

            if (x >= 0 && x < _singleCubeFaceEdgeDimension * 1)
                if (y >= _singleCubeFaceEdgeDimension * 1 && y < _singleCubeFaceEdgeDimension * 2)
                    cubemap.SetPixel(x, y, back.GetPixel(x, y % _singleCubeFaceEdgeDimension * 1));//Back

            if (x >= _singleCubeFaceEdgeDimension * 2 && x < _singleCubeFaceEdgeDimension * 3)
                if (y >= _singleCubeFaceEdgeDimension * 1 && y < _singleCubeFaceEdgeDimension * 2)
                    cubemap.SetPixel(x, y, front.GetPixel(x % (_singleCubeFaceEdgeDimension * 2), y % _singleCubeFaceEdgeDimension * 1));//Front

            if (x >= _singleCubeFaceEdgeDimension * 3 && x < _singleCubeFaceEdgeDimension * 4)
                if (y >= _singleCubeFaceEdgeDimension * 1 && y < _singleCubeFaceEdgeDimension * 2)
                    cubemap.SetPixel(x, y, right.GetPixel(x % (_singleCubeFaceEdgeDimension * 3), y % _singleCubeFaceEdgeDimension * 1));//Right

            if (x >= _singleCubeFaceEdgeDimension * 2 && x < _singleCubeFaceEdgeDimension * 3)
                if (y >= _singleCubeFaceEdgeDimension * 2 && y < _singleCubeFaceEdgeDimension * 3)
                    cubemap.SetPixel(x, y, down.GetPixel(x % (_singleCubeFaceEdgeDimension * 2), y % (_singleCubeFaceEdgeDimension * 2)));//Down
        }
    }
}
```

Figure 65 - Generarea Mapei Cubice

Cod nefolosit

In solutia programului putem regasi fisiere/clase/functii care nu mai sunt folosite. Toate aceastea au fost folosite cand am inceput prima data proiectul cu ideea de a face o proiectie din *n* camere. In libraria entities, putem regasi o clasa numita

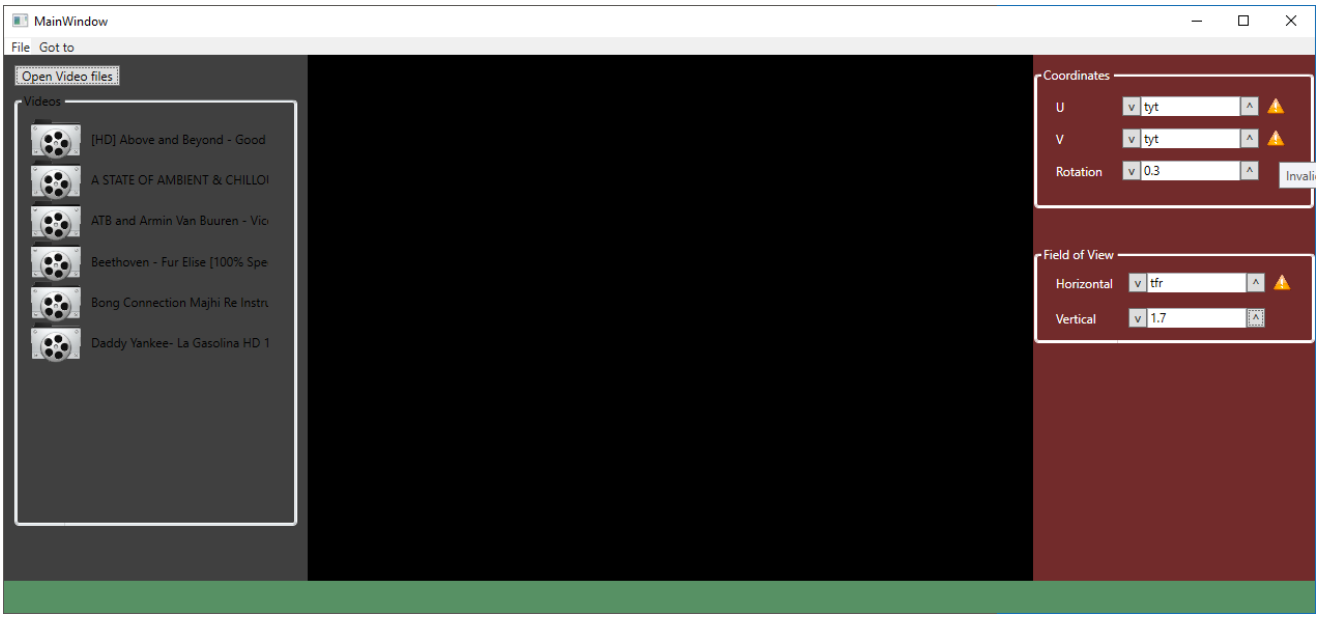


Figure 66 - Interfata UI, prima versiune

"CoordinatesValidator.cs". Scopul acestuia era sa fie folosita ca un atribut de validare pentru coordonatele unui video. Pentru *n* camere distribuite pe o sfera, fiecare trebuia sa aiba un set de coodonate: U, V, Rotation, HozFOV si VertFOV. Coordonatele U si V defineau coordonatele polare ale video-ului, "Rotation" definea rotatia camerei pe axa X a acesteia, iar HozFOV si VertFOV defineau cat de mare era unghiul de vedere al camerei atat de horizontala cat si pe verticala.

```
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field, AllowMultiple = false)]
sealed public class CoodinatesValidator : ValidationAttribute
{
    readonly int _min;
    readonly int _max;

    public CoodinatesValidator(int min, int max)
    {
        this._min = min;
        this._max = max;
    }

    public override bool IsValid(object value)
    {
        return (double)value <= _max && (double)value >= _min;
    }
}
```

```
public enum CoordType
{
    U,
    V,
    Rotation,
    Hoz_FOV,
    Vert_FOV
}
```

Figure 67 - Clasa ce trebuia sa se ocupe de validarea coodonatelor

Clasa “Coordinates.cs” din Entities reprezinta definitiile tuturor acelor coordonate impreuna cu atributele de validare.

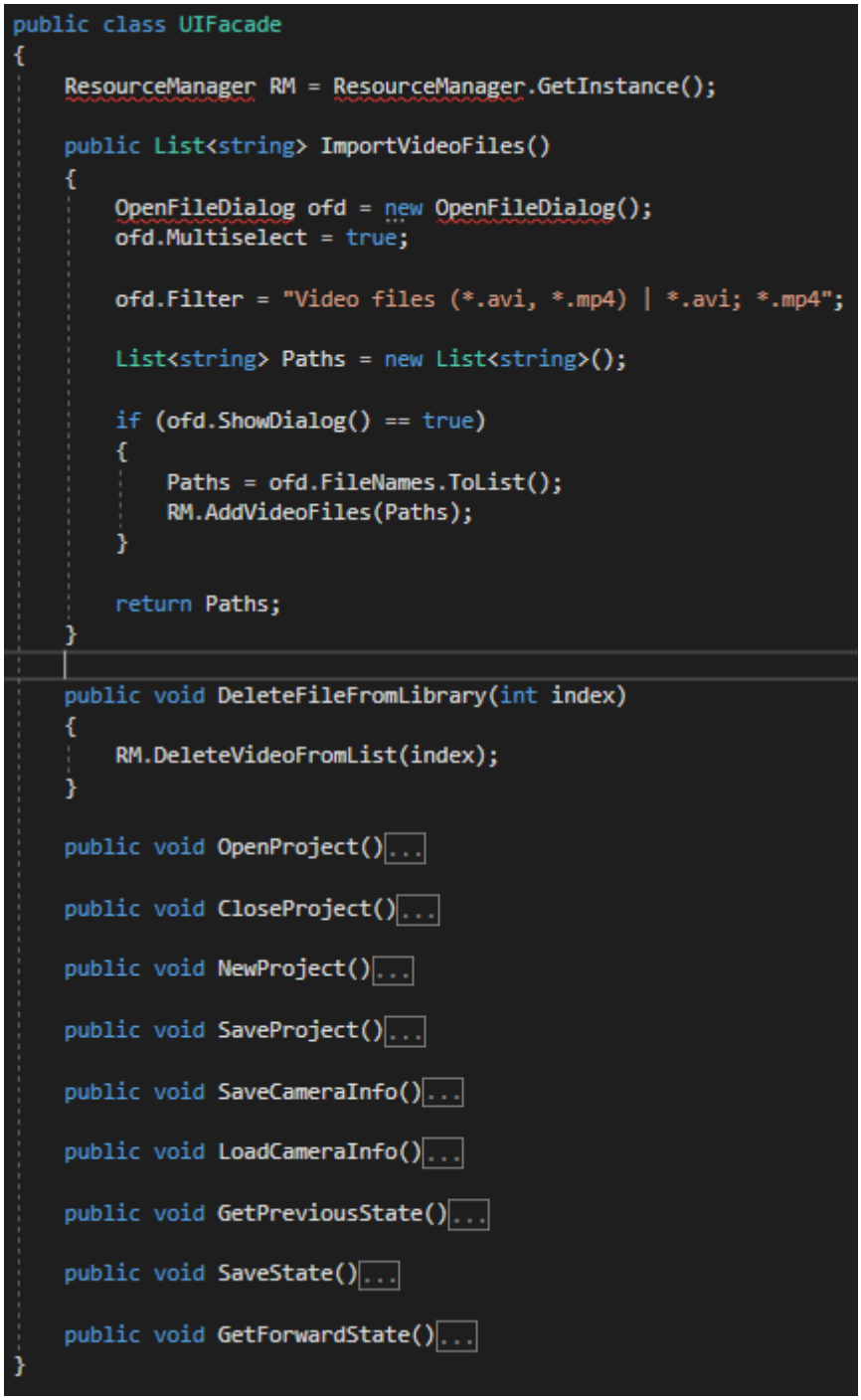
```
public class Coordinates
{
    [CoodinatesValidator(0, 1, ErrorMessage = "Invalid value")]
    public double U;
    [CoodinatesValidator(0, 1, ErrorMessage = "Invalid value")]
    public double V;

    [CoodinatesValidator(0, 360, ErrorMessage = "Invalid value")]
    public double Rotation;

    [CoodinatesValidator(0, 360, ErrorMessage = "Invalid value")]
    public double Hoz_FOV;
    [CoodinatesValidator(0, 360, ErrorMessage = "Invalid value")]
    public double Vert_FOV;
}
```

Figure 68 - Definitia coodonatelor

Lista de video-uri din partea stanga, puteau fi sterse atunci cand se apasa tasta “Delete”. Aceasta actiune stergeea de asemenea din “ResourceManager”. Logica de UI si clasa “ResourceManager” erau despartite printr-o fatada numita “Facade”. Rolul acesteia era de a simplifica apelul functiilor.

```
private void ScrollViewer_KeyDown(object sender, KeyEventArgs e)
{
    //if (e.Key == Key.Delete)
    //{
    //    for (int i = 0; i < VideoFilesList.Children.Count; i++)
    //    {
    //        var control = VideoFilesList.Children[i] as VideoFileUserControl;
    //        if (control.isSelected == true)
    //        {
    //            Facade.DeleteFileFromLibrary(i);
    //            VideoFilesList.Children.RemoveAt(i);
    //            eventManager.VideoDeletedFromList(i);
    //            break;
    //        }
    //    }
    //}
}
```

Figure 69 - Functia ce se ocupa cu stergerea unui video din lista



```
public class UIFacade
{
    ResourceManager RM = ResourceManager.GetInstance();

    public List<string> ImportVideoFiles()
    {
        OpenFileDialog ofd = new OpenFileDialog();
        ofd.Multiselect = true;

        ofd.Filter = "Video files (*.avi, *.mp4) | *.avi; *.mp4";

        List<string> Paths = new List<string>();

        if (ofd.ShowDialog() == true)
        {
            Paths = ofd.FileNames.ToList();
            RM.AddVideoFiles(Paths);
        }

        return Paths;
    }

    public void DeleteFileFromLibrary(int index)
    {
        RM.DeleteVideoFromList(index);
    }

    public void OpenProject()...

    public void CloseProject()...

    public void NewProject()...

    public void SaveProject()...

    public void SaveCameraInfo()...

    public void LoadCameraInfo()...

    public void GetPreviousState()...

    public void SaveState()...

    public void GetForwardState()...
}
```

Figure 70 - Clasa UI Facade cu diverse functii neimplementate

Clasa “Facade” trebuia sa implementeze functii pentru salvarea si crearea unui proiect. Am presupus ca utilizatorul are un “rig” (echipament) ale carui camera vor fi mereu indreptate in aceeasi pozitie, de aceea am vrut sa salvez acest echipament ca un XML folosind functia “SaveCameraInfo()” si incarcarea unui echipament folosind “LoadCameraInfo()”.

Deoarece utilizatorul are la dispozitie foarte multe controale de input, este usor ca acesta sa greseasca, de aceea, putem observa functii ca “GetPreviousState()”, “SaveState()” si “GetForwardState()”. La baza, aceste functia trebuiau sa implementeze “design pattern”-ul

numit “Memento”, sau functionalitatea de “Undo” si “Redo” (anula si a reface) folosind combinatia de taste CTRL+Z si CTRL+Y.

Functiile “OpenProject()”, “CloseProject()”, “NewProject()” si “SaveProject()” aveau rolul de a salva pe disc starea curenta a programului. In programul final nu am mai implementat aceasta functionalitate deoarece utilizatorul are cativa pasi clari si simplu de urmat pentru a ajunge la rezultatul final.

Rezultatul final

Video-ul final poate fi deschis foarte usor folosind VLC player pentru a testa rezultatul si calitatea. Observatie: acest video este special dedicat unor dispozitive speciale cum ar fi cum ar fi “HTC Vive” sau “Oculus Rift”.

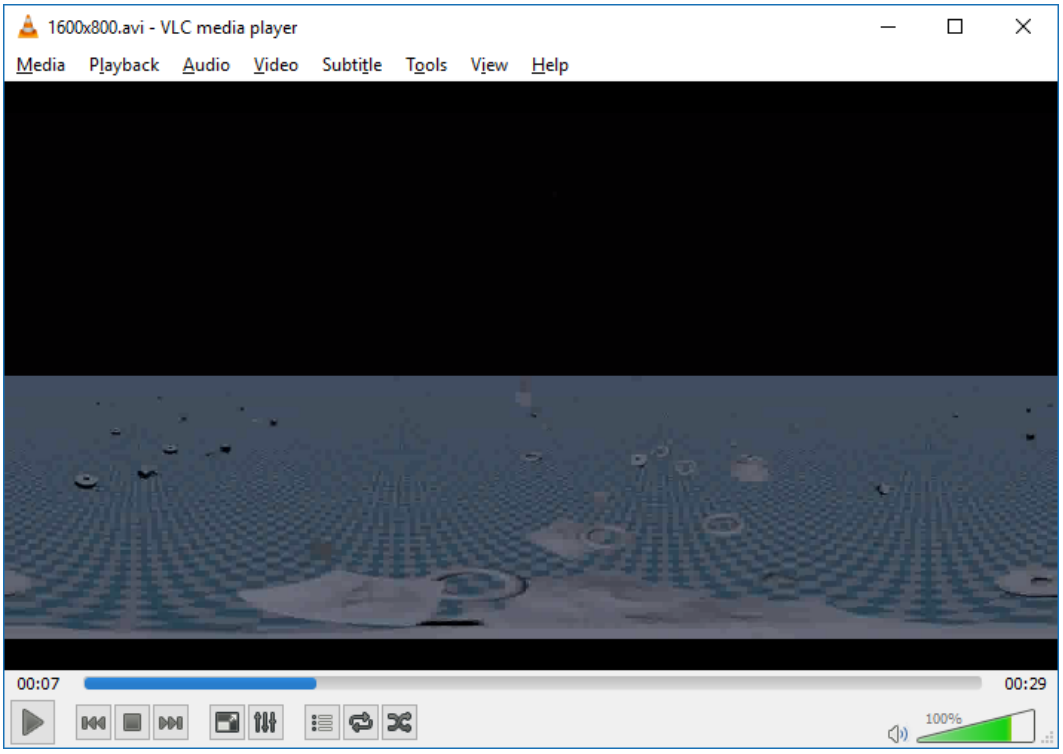


Figure 71 - VLC player cu video-ul randat de program

Figure 72 - Headset "VR"

Aceste dispozitive pot face urmatorul lucru: avand un cadru ce reprezinta o proiectie Equirectangulara, o pot mapa pe o sfera folosind algoritmul “UV mapping”. Algoritmul este sens invers fata de cel descris la capitolul I.

Folosind serviciul video Youtube si cu ajutorul unui telefon care suporta “VR”, se poate urmari video-ul intr-o maniera usoara si simpla.

Concluzii

Avand in vedere faptul din cei in ce mai multi oameni descopera cat de usor este sa faci un video atat calitativ cat si cantitativ si sa il postezi pe diverse retele sociale, programul facut de mine se incadreaza usor in arsenalul de software-uri de editare si procesare video.

Aplicatia mea nu se opreste aici. Urmatorul pas ar fi ca aplicatia sa fie capabila de a crea o panorama 360 din mai mult de 6 video-uri. Pentru acest lucru ar trebui facute o serie de investigari in domeniul Homografiei, domeniu ce se ocupa cu lipirea imaginilor (“image stiching”). Prima varianta a aplicatiei a implicat foarte multe inputuri din partea utilizatorului. As vrea sa automatizez acest proces de adaugare a video-urile, sa creez un sistem inteligent pentru detectia coordonatelor fiecarei camere, utilizatorul facand un efort minim pentru a ajunge la rezultatul final dorit.

In productia cinematografica, se folosesc programe speciale cum ar fi “Adobe After Effects” sau “Nuke” sau chiar “Cinema 4D”, progarme ce suporta “plugin”-uri, sau assembly-uri capabile sa faca doar un singur lucru, iar utilizatorul sa foloseasca in mod direct rezultatul fara sa isi schimbe mediul de lucru. Programul meu ar putea fi portat intr-un plugin usor, deoarece codul care face lucrurile cele mai importante au fost scrise in librarii sau assembly-uri, singurul lucru de facut fiind crearea unui “wrapper” (invelis) in jurul acestui assembly pentru a fi suportat de catre mediul de dezvoltare al software-ului destinatie.

Programul de fata este unul de tip Desktop, deoarece se incadreaza mai bine pentru ceea ce face si cum e utilizat. O interfata web facuta pentru telefoanele mobile i-ar ajuta si pe cei care nu se afla mereu in fata unui calculator sa isi distribuie experientele intr-un format 360. Portarea acestei aplicatii pe o aplicatie web s-ar face usor datorita celor doua assembly-uri “PreviewRenderer” si “RenderEngine” scrise separat inca de la inceput.

Mereu am fost inspirat de puterea unui software open source. Spre exemplu “VLC player”, un video player “open source”, a ajuns cel mai puternic software pentru redarea fisierelor multimedia. O idee cum este a mea de creeare a video-urilor 360 ar putea sa ajunga cel mai bun software in acest domeniu al cinematografiei, la fel ca “VLC player”.

Bibliografie

[1] Map Projections – A Working Manual, by John P. SNYDER

[2] Introduction to Map Projections - http://www.microimages.com/documentation/Tutorials/project.pdf

[3] https://en.wikipedia.org/wiki/UV_mapping

[4] https://en.wikipedia.org/wiki/Equirectangular_projection

[5] https://ro.wikipedia.org/wiki/Coordonate_polare

[6] https://en.wikipedia.org/wiki/Cylindrical_coordinate_system

[7] https://code.facebook.com/posts/1126354007399553/next-generation-video-encoding-techniques-for-360-video-and-vr/

[8] http://paulbourke.net/miscellaneous/cubemaps/

[9] https://arxiv.org/ftp/arxiv/papers/1011/1011.3189.pdf

[10] https://en.wikipedia.org/wiki/Homography_(computer_vision)

Anexe

Anexa 1

Imagini preluate din surse externe

- Fig. 1 = https://i.stack.imgur.com/Q1SO4.png
- Fig. 2 = https://en.wikipedia.org/wiki/Mercator_projection#/media/File:Mercator_projection_SW.jpg
- Fig. 3 = https://upload.wikimedia.org/wikipedia/commons/8/83/Equirectangular_projection_SW.jpg
- Fig. 72 = http://cdn.mos.cms.futurecdn.net/d30ae6fa43191d61b97dabb2b62471ed-480-80.jpg
- Fig. 6 = http://r4r.co.in/c1/01/tutorial/csharp/image/num.JPG
- Fig. 71 = http://www.videolan.org/vlc/

Anexa 2

Cod preluat din surse externe

- Proiectia Equirectangulara = https://stackoverflow.com/questions/34250742/converting-a-cubemap-into-equirectangular-panorama
- Transformarea din “Bitmap” in “BitmapImage” = https://stackoverflow.com/questions/6484357/converting-bitmapimage-to-bitmap-and-vice-versa

Anexa 3

Librarii ce nu imi apartin

- Libraria AFORGE = http://www.aforgenet.com/

Anexa 4

Modelul 3D

- https://www.turbosquid.com/3d-models/free-base-heads-3d-model/721696

Anexa 5

Al meu

- Interfata UI
- Codul din spatele interfetei
- Imaginile folosite
- Codul din spatele celor 2 librarii: “PreviewRenderer” si “RenderEngine”
- Controalele customizate
- Codul care nu a fost trecut mai sus