Design of Applications, Service and Systems

Project name:
Neuro-load reduction on video stream for kids

Vahid Farjood Chafi
Design Of Applications, Services and Systems
M.S. Computer and Automation Engineering

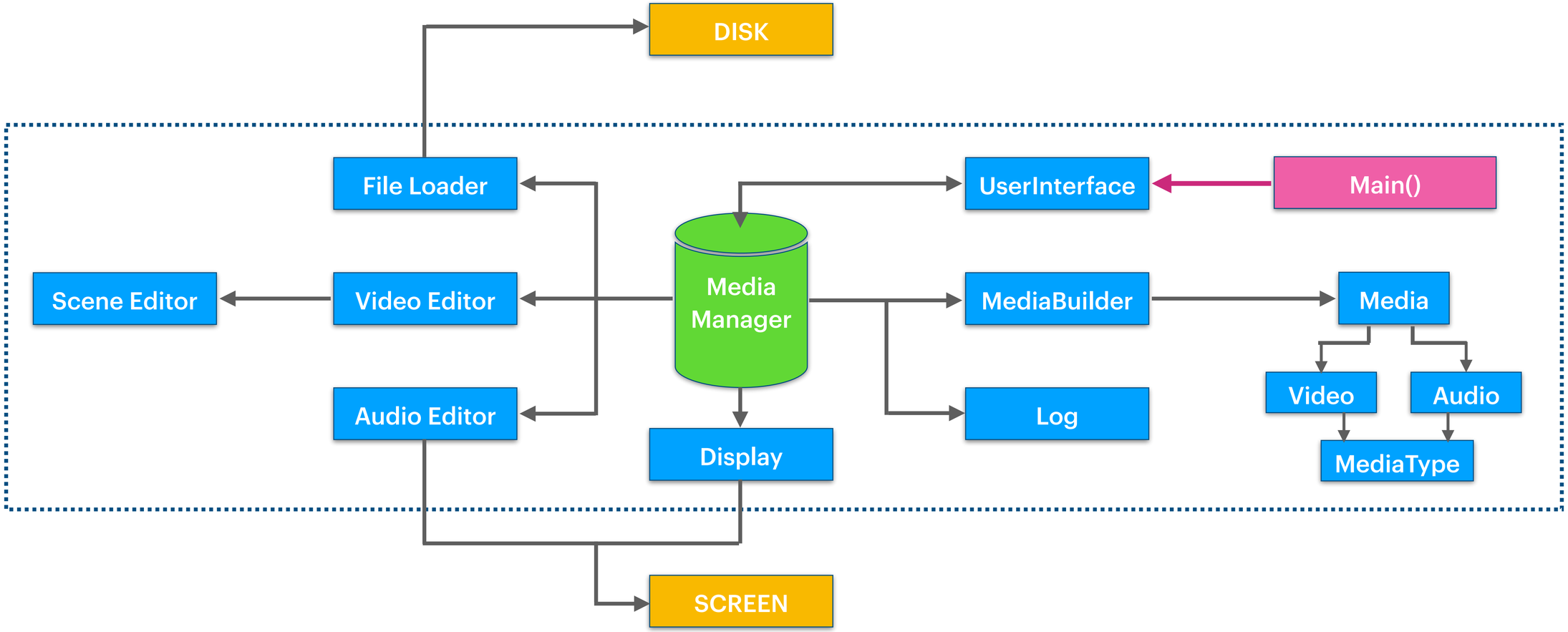
University of Siena

Outline

Given a video stream with 'n' scene transition per minute, how can we reduce number of scene in the stream?

$$result = n/i$$ *where i=1,2,3,... (max scene change per minute/second defined by user)*

Architecture



Vahid Farjood Chafi
Design Of Applications, Services and Systems
M.S. Computer and Automation Engineering

University of Siena

Media Manager

MediaManager is the core component of the application which connects and controls all other parts.

Features:

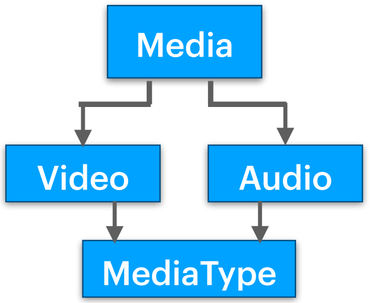
- Integrates and connects all components together.
- Provide enough methods to interact with different part of the application.
- COPY constructor and assignment are protected and explicitly deleted since there must be only one owner of media manager.

Media Class

Media object is basically a wrapper over video and audio classes in which they are derived classes from MediaType base class.

Features:

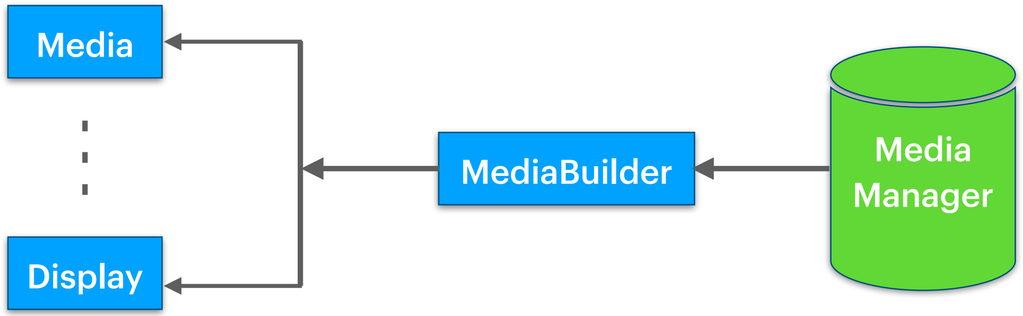
- Video/Audio entities are derived classes from MediaType base class.
- They have some common attributes such as *stream_index or duration()* from MediaType.
- They also have their own attributes and buffer.



Media Builder

Builder Design Pattern
All object creational procedures are done by this builder class.

Features:
- Create objects during runtime based on demand.
- Classes delegate object creation to the Builder class instead of creating the objects directly.

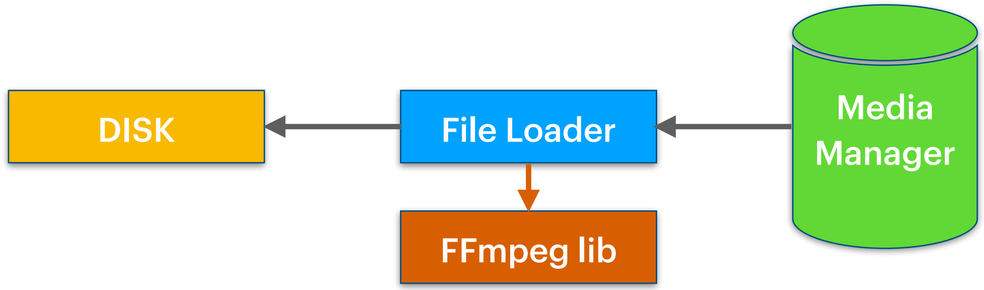


File Loader

FileLoader is used to load and decode media from disk and it uses FFmpeg library.

Features:

- Receive video/audio objects as input from media manager.
- Read media files from disk.
- Get all the necessary informations of the stream and fill them to video/audio object.
- Send video/audio objects as an output to media manager.
- Ability to clean up all related contexts.

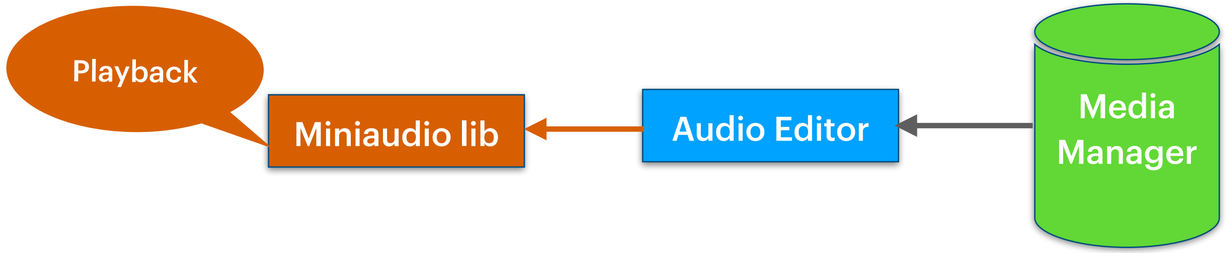


Audio Editor

This component is used to manage and control audio stream specially for playback:

Features:

- Receive audio objects as input from media manager.
- Uses miniaudio library for playing the audio track.

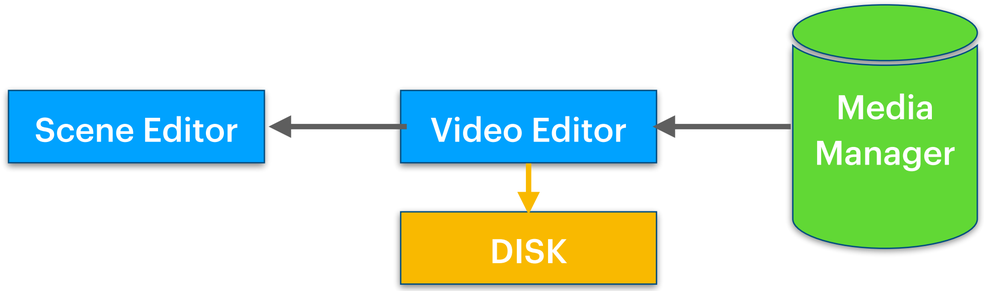


Video Editor

This component is used to manage video stream manipulation tasks including scenes which is under the SceneEditor class:

Features:

- Receive video objects as input from media manager.
- Provide enough methods to edit and manipulate the video stream like splitting.
- Uses SceneEditor class to manipulate scenes.
- Uses OpenCV for some video manipulations including saving video to the disk.
- Send back video objects as an output to the media manager.

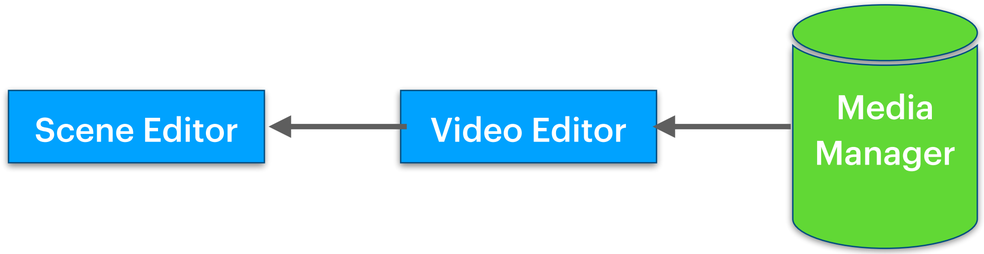


Scene Editor

This component is used under VideoEditor class and it provide all the important part of scene manipulation tasks:

Features:

- Receive video buffer from VideoEditor as an input.
- Provide methods to detect automatically all the scenes in video stream.
- Provide methods to fill missing frames.
- Send back video buffer as output to the VideoEditor.



Scene Editor

Scene Detector:

- Calculate all frames average in term of color density
- Compare each frame average with its next frame with respect to a certain threshold.
- If the difference is above the threshold there is a scene transition and we save the index of it in a new vector called *scene_indexes*.

```
void SceneEditor::sceneDetector(std::vector<uint8_t*>& video, int const& width, int const& height, int
const& threshold)
{
   std::vector<double> framesAverage;
   frameAverage(video, framesAverage);    // function to calculate each frame average
   for(int i = 5; i < framesAverage.size(); i++)
   {
      int diff = std::abs(framesAverage[i-1] - framesAverage[i]);
      if(diff > threshold)
      {
         scene_indexes.emplace_back(i);
       }
   }
}
```

Scene Editor

Scene Replacer:

- Receive a video buffer and an option for filling the missing frame.
- Loop over the video buffer based on the number of scenes.
- For each scene, if the overall number of frames in that scene takes less than one second, then update those frames with new frames based on the option provided by user.

```
void SceneEditor::sceneReplacer(std::vector<uint8_t*>& video, std::string const& option, int const& fps)
{
   for (int curr_scene = 2; curr_scene < scene_indexes.size(); curr_scene++)
   {
      if((num_of_frame_in_scene < fps) && (!previous_scene_was_reduced))
      {
         // options to fill the removed frames:
         ...
      }
   }
}
```

Scene Editor

Black or White:

For each frame of the removed scene, replace it's pixels with black\white color by passing the frame to the *updateFrame* function with proper alpha and beta value:

```
if(option=="black")
{
   for(int i = scene_indexes[curr_scene-1]; i < scene_indexes[curr_scene]; ++i)
   {
      video[i] = updateFrame(video[i], 0, 0);
   }
}

if(option=="white")
{
   for(int i = scene_indexes[curr_scene-1]; i < scene_indexes[curr_scene]; ++i)
   {
      video[i] = updateFrame(video[i], 0, 255);
   }
}
```

Scene Editor

UpdateFrame:

For each frame:

- Loop over all the pixels of that frame.
- Change each pixel based on alpha\beta value

```
uint8_t* SceneEditor::updateFrame(uint8_t* frame, double const& alpha, int const& beta)
{
   for( int x = 1; x < m_height; x++ ) {
      for( int y = 0; y < m_width; y++ ) {
         for( int c = 0; c < 3; c++ ) {
            frame[x*m_width*4 + y*4 + c] = frame[x*m_width*4 + y*4 + c] * alpha + beta;
         }
       }
    }
   return frame;
}
```

Scene Editor

Freeze the removed frame, freeze the last frame, freeze the next frame:

```
if(option==“freeze") // freeze the removed frame:
{
   for(int i = scene_indexes[curr_scene-1]; i < scene_indexes[curr_scene]; ++i)
   {
      video[i] = video[scene_indexes[curr_scene-1]];
   }
}

if(option==“last-frame") // show the last frame of the scene before the remove one:
{
   for(int i = scene_indexes[curr_scene-1]; i < scene_indexes[curr_scene]; ++i)
   {
      video[i] = video[last_scene_frame];
      last_scene_frame = i;
   }
}

if(option==“next-frame") // show the next frame of the scene after the remove one:
{
   for(int i = scene_indexes[curr_scene-1]; i < scene_indexes[curr_scene]; ++i)
   {
      video[i] = video[scene_indexes[curr_scene]];
   }
}
```

Scene Editor

Blur the last frame to next frame:

Show the blurred last frame of the scene before the remove one, and then show the blurred of the next frame after removed one:

```
if(option=="blure-last-to-next")
{
   for(int i = scene_indexes[curr_scene-1]; i < scene_indexes[curr_scene]; ++i)
   {
      if(i < ((scene_indexes[curr_scene-1] + scene_indexes[curr_scene])/2))
         video[i] = updateFrame1(video[i], video[last_scene_frame] );
      else
         video[i] = updateFrame1(video[i], video[scene_indexes[curr_scene]]);

      last_scene_frame = i;
   }
}

uint8_t* SceneEditor::updateFrame1(uint8_t* frame1, uint8_t* frame2)
{
      int size = m_width*m_height*4; // 3.686.400
      for(int i=4; i < size-4; i+=4){
         int b_avg = (frame2[i-4] + frame2[i]   + frame2[I+4])/3; // blue
         int g_avg = (frame2[i-3] + frame2[i+1] + frame2[I+5])/3; // green
         int r_avg = (frame2[i-2] + frame2[i+2] + frame2[I+6])/3; // red
         frame1[i] = b_avg;
         frame1[i+1] = g_avg;
         frame1[i+2] = r_avg;
      }
   return frame1;
```

Scene Editor

Fade the removed scene to the next scene:

Fade the last frame of the remove scene to the next frame after removed scene:

```
if(option=="fade")
{
   for(int i = scene_indexes[curr_scene-1]; i < scene_indexes[curr_scene]; ++i)
   {
      video[i] = updateFrame2(video[i], video[scene_indexes[curr_scene]], num_of_frame_in_scene);
      if(num_of_frame_in_scene < 1)
         continue;
      --num_of_frame_in_scene;
   }
}

uint8_t* SceneEditor::updateFrame2(uint8_t* frame1, uint8_t* frame2, int const& step)
{

   for( int x = 1; x < m_height; x+=step ) {
      for( int y = 0; y < m_width; y+=step ) {
         for( int c = 0; c < 3; c++ ) {
            frame1[x*m_width*4 + y*4 + c] = frame2[x*m_width*4 + y*4 + c];
         }
         }
    }
   return frame1;
}
```

Scene Editor

Fade-in the last frame to fade-out the next frame:

Fade-in the last frame of the scene before the remove one to fade-out the next frame of the scene after removed one:

```
if(option=="fade-last-to-next")
{
   for(int i = scene_indexes[curr_scene-1]; i < scene_indexes[curr_scene]; ++i)
   {
      video[i] = updateFrame1(video[i], video[last_scene_frame] );
      video[i] = updateFrame2(video[i], video[scene_indexes[curr_scene]], num_of_frame_in_scene);
      if(num_of_frame_in_scene < 1)
         continue;
      --num_of_frame_in_scene;
      last_scene_frame = i;
   }
}
```

Log

A complete log service for all debugging purposes in the entire part of the application.

Features:

- Singleton class.
- Use of mutex in order to avoid data races.
- Different log level.
  - *Info, warning, error, buffer, trace, debug*
- Different log mode.
  - *Console, file*
- Provide methods to change log level and mode.
- Provide direct interface using Macros.
- Ability to change the configuration during run time.

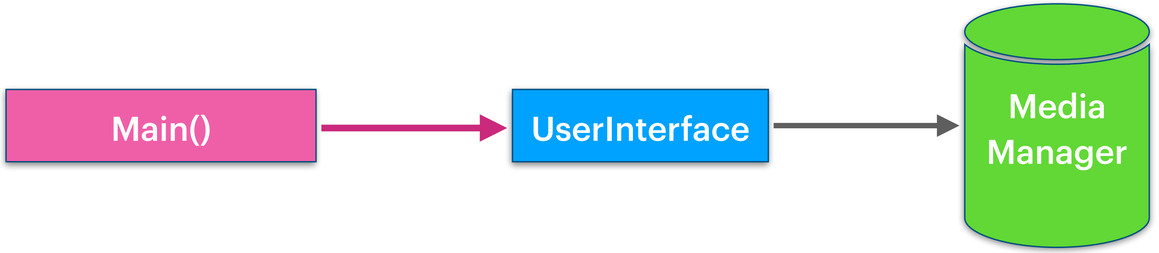
```
[10] [Mon Jan 10 12:28:21 2022] [DEBUG]:   ["MediaManager:  Constructor of the object: 0x7ff7b9f177b0"]
[11] [Mon Jan 10 12:28:39 2022] [DEBUG]:   ["FileLoader:    Constructor of the object: 0x600002530000"]
[12] [Mon Jan 10 12:28:39 2022] [TRACE]:   ["MediaManager:  Media has been loaded from: ../media/trailer/Trailer.mp4"]
[13] [Mon Jan 10 12:28:43 2022] [INFO]:    ["MediaManager:  Video stream has been decoded."]
[14] [Mon Jan 10 12:28:43 2022] [DEBUG]:   ["SceneEditor:   Constructor of the object: 0x600002738c00"]
[15] [Mon Jan 10 12:28:43 2022] [DEBUG]:   ["VideoEditor:   Constructor of the object: 0x600002738bb0"]
```

User Interface

A very simple command-line user interface to interact with the application. This is the only class that you need to include in your main.cpp file.

Features:

- Uses Labels to jump into different state of the application.
- Provide different helps for user to give more information of how the program works.
- Provide methods to change log configuration.
- Syntax error detection and create also an error sound if user types a wrong command.



Conclusion

Pros:

- Scaleability and modularity of the design which allows to add some more features easily.
- Use of some standard design patterns for different purposes (builder, singleton)
- A complete logging service which covers all the need for debugging purposes.
- Implementation of the User Interface for interacting with the application.

Cons:

- Algorithm to detect scenes is not efficient and it would be better to use some more computer vision techniques like object detection to find a scene transition.
- Need to design a better user interface since this user interface is not scalable as the program grows.
- There is no save method to store the edited video stream with its audio track.
- Need to devise a good method to synchronize video with audio track after editing them.

Thank you for your attention.