2.FEASIBILITY STUDY

PRIOR WORKS

There are many approaches to synthesizing videos of dynamic scenes. One approach that has garnered a lot of attention is video texture , which reuses frames to generate a seamless video of arbitrary length. Video textures work by figuring out frames in the original video that are temporally apart but visually close enough, so that jumping between such frames (via a first-order Markov Chain model) appears seamless. This work was extended to produce video sprites , which permit high-level control of moving objects in the synthesized video. Unlike videos, the ordering of our input stills may not be 1D. Thus, we can only use partial orders as reference dynamics. In addition, we adopt a second-order Markov Chain model for generating image sequences, rather than the first-order Markov Chain model in . While the video texture paper mentioned used independent regions (IARs in our case), it did not address the issue of local region decomposition. As we demonstrate in our paper, local (IAR) decomposition is another important operation required to produce seamless animation.

Kwatra et al. further extended video textures by recomposing different frames with graph cuts instead of reshuffling the complete frames. Agarwala et al. created panoramic video textures using min-cut optimization to select fragments of video that can be stitched together both spatially and temporally. They also manually partitioned the scene into static and dynamic regions. Sun et al. developed a video-input driven animation system to extract physical parameters such as wind speed from real videos. These parameters are then used to drive the physical simulation of synthetic objects.

Many approaches rely on user input to specify motion in the synthesized video. Bhat et al. , for instance, synthesized flow-based videos by analyzing the motion of textured particles in the input video along user-specified flow lines and synthesizing seamless video of arbitrary length by enforcing temporal continuity along other user specified flow lines. Litwinowicz and Williams used key frame line drawings to deform images to create2D

animation. Treuille et al. also used key frames to control the smoke simulation. In , video sequences of a person's mouth were extracted from a training sequence of the person speaking and then reordered in order to match the phoneme sequence of a new audio track. Aoki et al. combined physically-based animation and image morphing techniques to simulate and synthesize plants. Chuang et al.'s system allows the user to animate a single image. Here, all motion is assumed to be caused by wind. In addition, the user has to manually segment the image layers and specify the motion model for each layer.

Some approaches for synthesizing dynamic scenes are based on more mathematically rigorous analysis. For example, Wang and Zhu modeled the motion of texture particles in video using a second-order Markov chain. Soatto et al. applied nonlinear dynamic systems to model dynamic textures and borrowed tools of system identification to capture the essence of the dynamic textures. Szummer and Picard built a spatio-temporal autoregressive model for temporal textures.

Human perception has also been considered in producing dynamic textures. Freeman et al. applied quadrature pairs of oriented filters to vary the local phase in an image to give the illusion of motion. Paintings can also be illuminated by sequentially timed lights to create the illusion of motion, e.g., the kinetic waterfall .

GRAYSCALE CONVERSION

To convert any color to a grayscale representation of its luminance, first one must obtain the values of its red, green, and blue (RGB) primaries in linear intensity encoding, by gamma expansion. Then, add together 30% of the red value, 59% of the green value, and 11% of the blue value (these weights depend on the exact choice of the RGB primaries, but are typical). Regardless of the scale employed (0.0 to 1.0, 0 to 255, 0% to 100%, etc.), the resultant number is the desired linear luminance value; it typically needs to be gamma compressed to get back to a conventional

grayscale representation. The image given as input is represented as pixels and the red, green and blue component of each pixel is change to . 299*red+.587*green+.114*blue because the red, green and blue components of grayscale picture has values 299, 587 and 114 respectively. Gray scale filtering is in reference to the color mode of a particular image. A gray scale image would, in layman's terms, be a black and white image, any other color would not be included in it. Basically, it's a black and white image, the colors in that image, if any will be converted to corresponding shade of gray (mid tones between black and white) thus, making each bit of the image still differentiable.

INVERSION

Inversion means changing the component value of each pixel to with a difference of 255. This is so simple that it doesn't even matter that the color components are out of order. It is just taking the opposite color of the current component. That is for example if the color component is 00 then the opposite we get is FF (255-0). It is very simple - it just adds or subtracts a value to each color. The most useful thing to do with this filter is to set two colors to -255 in order to strip them and see one color component of an image

BRIGHTNESS ALTERING

Brightening images are sometimes needed, it's a personal choice. Sometimes printing needs a lighter image than viewing. It is done just by adjusting the color components as per the user requirement. The input ranges between -255 and 255. The input value is added to each pixel component of the input image to change brightness. Then the brightness will increase or decrease according to the given value.

ZOOM IN AND ZOOM OUT

This is resizing the width and height of the image without affecting any pixels of the image so that it does not affect the resolution of the image. The image is represented as bits. The input value is multiplied with the width and also with the height of this pixels in order to zoom the image. Based on original image and input value the image will be zoom in or zoom out. The original image is considered to be 100% zoom. So a value less than 100 will zoom in the image while a value greater than 100 will zoom out the image.

WATERMARKING

Adding a visible watermark is a common way of identifying images and protecting them from unauthorized use online. A watermark is a visible embedded overlay on a digital photo consisting of text, a logo, or a copyright notice. The purpose of a watermark is to identify the work and discourage its unauthorized use. Though a visible watermark can't prevent unauthorized use, it makes it more difficult for those who may want to claim someone else's photo or art work as their own. An image is watermarked by drawing a string with the input attributes on the image. The text to be drawn, it’s font, type, color and size can be specified.

MODULE 2

This module consists of the comparison and sorting functions. This can be illustrated as

The input images are compared and their pixel difference is stored. This pixel difference value will help in sorting. The first image in output sequence will be specified. It is based on this first image the other images are sorted. For the sorting purpose any of the sorting techniques can be used. In our system a simple selection sort is used for the purpose.

COMPARISON

The two input images will be first represented in pixels. Starting from the top left corner of both images ,each pixel is compared. The pixel difference is saved . This pixel difference is using for sorting.

SORTING

The system first builds an array of input images. We can make processing on these saved images. The image to be appear as the first image in animation is assigned by us. The remaining images are arranged on basis of the first image.

The next module is the image comparison part. The images in image array is compared and get the number of pixels they differ from each other. This comparison take two images from image array and compare their pixels starting from the first pixel on top left corner.

Next is the sorting phase. It is based on the comparison. The first image to appear in animated output is given as first image in array. Then all other images are compared with first image and the second image is set. Then by comparing the remaining images with second image the third image is fix and this process is repeated for finding the positions of all remaining images. The selection sort algorithm is using for the purpose.

As in selection sort algorithm two loops are used. The first image is already is fixed by user. It is done by first loop. In the inner loop all other images are compared with the first image and their pixel difference to first image is saved in a variable. The one with lowest pixel difference is taken as second image. Then in outer loop the second image is set. Then in inner loop remaining images are compared with second image and the pixel difference is saved. The one with lowest pixel difference is taken as third image. This process continues and all image positions are set.

VIDEO MAKING

The video making part is accomplished using a byte scout image to video library. The slide duration for each image is set in this and an

animated output is given by using methods in byte scout image to video library.

4.SYSTEM SPECIFICATION

This project can run on windows xp or 7 since it uses Microsoft Visual Studio .net 2010 as front end. The coding language used is visual C# .NET. It is used because it is an elegant, simple, type safe, object oriented language. It has the capability to build durable system-level components by virtue of the following features: It is a Full COM/Platform support for existing code integration. It has robustness through garbage collection and type safety. Also security is provided through intrinsic code trust mechanisms. It also provide full support of extensible metadata concepts.

Since our system is based on images .net and c# are the proper back end and front end. The application uses the basic Windows Forms application. Our system have handled the images with a separate class called Image Handler in which all the image related operations are done including Saving, Graphics related operations. The Functionality includes getting image information, zooming, color filtering, brightening, contrasting, gamma filtering, gray scale filtering, invert filtering, resizing with full resolution, rotating and flipping, cropping and inserting text, any other image and some geometric shapes. Scrolling is achieved in the standard manner. The Paint method uses the Auto Scroll Position property to find out our scroll position, which is set by using the Auto Scroll Min Size property.

Color filters are very simple - it just adds or subtracts a value to each color. The most useful thing to do with this filter is to set two colors to -255 in order to strip them and see one color component of an image. For example, for red filter, keep the red component as it is and just subtract 255 from the green component and blue component. Brightening images are sometimes needed, it's a personal choice. Sometimes printing needs a lighter image than viewing. It is done just by adjusting the color components as per the user requirement. The input ranges between -255 and 255. Gray scale

filtering is in reference to the color mode of a particular image. A gray scale image would, in layman's terms, be a black and white image, any other color would not be included in it. Basically, it's a black and white image, the colors in that image, if any will be converted to corresponding shade of gray (mid tones between black and white) thus, making each bit of the image still differentiable.

The .net features like windows form application make it easy to design the forms and developing the project. The file dialogues like open file dialogue and save file dialogue make it easy to implement loading and saving of files from and to locations in the system. The menu strip helps to make the menu bar and coding its functionalities. The tools like button , label ,drop down list, combobox, checkbox, listbox ,etc helps in designing the forms . The pictures can be easily loaded using picturebox . Panel makes it feasible to view a number of pictures simultaneously.

It also inter-operates with other languages, across platforms, with legacy data ,by virtue of the following features: It has full interoperability support through COM+ 1.0 and .NET Framework services with tight library-based access. It provides XML support for Web-based component interaction. It is a versioning to provide ease of administration and deployment.

The features of VS .NET that make us more productive are
1.Drag and Drop design
2.IntelliSense features
3. Syntax highlighting and auto-syntax checking
4.Excellent debugging tools
5.Integration with version control software such as Visual Source Safe (VSS)
6.Easy project management

.NET represents an advanced new generation of software that will drive the Next Generation Internet. Its purpose is to make information available any time, any place, and on any device.

The three kinds of assemblies that you can createwith C# are the following.· Console applications, GUI applications, Libraries of Types

As components are added to the Form, Visual Studio assigns default names to each one. It is via these names that any C# code will interact with the user interface of the application. For this reason it is important to specify meaningful names which identify the component when referenced in the C# source code. It is recommended, therefore, that the default names provided by Visual Studio be replaced by more meaningful ones. These all helps us in developing our project easily.

The important thing to remember about assembliesisthatthey are not source-code. They are compiled binaries that can be executed directly. The relationship between C# and the .NET Framework is somewhat unique. First, C# is not the only language that can be used to write .NET Framework applications (calledManaged Applications). Second, .NET or managed, applications run in native machine-language and are not interpreted. Third, C# or managed applications do not run in a sandbox.

C# .NET allows you to draw straight to a form or form objects. You do all the drawing with inbuilt Graphic objects. Drawing in C# is achieved using the *Graphics Object*. The Graphics Object takes much of the pain out of graphics drawing by abstracting away all the problems of dealing with different display devices and screens resolutions. The C# programmer merely needs to create a Graphic Object and tell it what and where to draw. Graphics Object for a component and then drawing on that component does not create persistent graphics. In fact what will happen is that as soon as the window is minimized or obscured by another window the graphics will be erased. For this reason, steps need to be taken to ensure that any graphics are persistent. Two mechanisms are available for achieving this. One is to repeatedly perform the drawing in the*Paint()* event handler of the control (which is triggered whenever the component needs to be redrawn), or to perform the drawing on a bitmap image in memory and then transfer that image to the component whenever the *Paint()* event is triggered.

Most of the controls we use to create our applications are defined in the .NET Framework and, each is based on a particular class. To provide them with basic and common characteristics, the visual Windows

controls of the .NET Framework based on a class called Control which is defined in the System.Windows.Forms namespace of theSystem.Windows.Forms.dll assembly are used. Based on this, the characteristics common to .NET Framework graphical controls are accessed and managed from one point, then inherited by those controls.

AviManger manages the streams in an AVI file. The constructor takes the name of the file and opens it. Close closes all opened streams and the file itself. You can add new streams with AddVideoStream and AddAudioStream. New video streams are empty, Wave streams can only be created from Wave files. After you have created an empty video stream, use the methods of VideoStream to fill it. AviManger manages the streams in an AVI file

With this workaround, we are able to call AVI Save Options and (later on) AVISaveV in C#. Now, the new stream can be filled with image frames using AddFrame.

BytescoutImageToVideo library has inbuilt functions in it to produce a video from a series of images

The hardware needed include Pentium IV processor with 32 bit system bus , 512 MB RAM, 40 GB hard disk and SVGA color display.

SOFTWARE SPECIFICATION

Operating system	:Windows 7
Front End	:Microsoft VisualStudio .NET 2010
Coding Language	:Visual C# .NET with GDI+Components.

HARDWARE SPECIFICATION

Processor	: Pentium IV
System Bus	: 32 Bit
RAM	: 512 MB
HDD	: 40 GB
Display	: SVGA Color
Key Board	: Window/Linux compatible

5.IMPLEMENTATION

In this project, we describe a system that allows the user to quickly and easily produce a compelling-looking animation from a small collection of high resolution stills.

As the first phase we implemented a image processing widget using c#. In this we can input a number of images which are the inputs to our proposed system and can do some processing on these images.

Using this widget we can load images onto the windows and can do operations like changing brightness, contrast, converting to black and white or grayscale inversion and zoom-in/zoom-out. The image loading function is implemented as

```
File_Load(object sender, System.EventArgs e)
{
if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
           this.CurrentImage = Image.FromFile(openFileDialog1.FileName);
           Image img = (Image)this.CurrentImage.Clone();
           pictureBox1.Image = (Image)CurrentImage.Clone();
           g = Graphics.FromImage(img);
           b = new Bitmap(img);
}
```

Using above functions all images in the format .jpeg or .bmp can be loaded to the canvas(refer appendix 1). It is set as the initial file directory to be loaded as the c directory. We can choose images from any location in our system using this function. The selected image will auto fit to the canvas.

To save an image ,it is using the following function

```
File_Save(object sender, System.EventArgs e)
{
SaveFileDialog saveFileDialog = new SaveFileDialog();
        saveFileDialog.InitialDirectory = "c:\\";
          saveFileDialog.Filter = "Bitmap files (*.bmp)|*.bmp|Jpeg files (*.jpg)|
*.jpg|All valid files (*.bmp/*.jpg)|*.bmp/*.jpg";
        saveFileDialog.FilterIndex = 1;
        saveFileDialog.RestoreDirectory = true;
        if (DialogResult.OK == saveFileDialog.ShowDialog())
        {
           this.CurrentImage = pictureBox1.Image;
           Image img = (Image)this.CurrentImage.Clone();
           img.Save(saveFileDialog.FileName);
           //m_Bitmap.Save(saveFileDialog.FileName);
        }
```

A dialog box will be loaded and we can specify the location to save the image. The exit function is defined as follows

```
private void File_Exit(object sender, System.EventArgs e)
                {
                        this.Close();
                }
```

Zoom function is defined by

```
this.AutoScrollMinSize =new Size ((int)(m_Bitmap.Width * Zoom), (int)
(m_Bitmap.Height * Zoom));
```

It is implemented as follows

```
private void OnZoom25(object sender, System.EventArgs e)
                {
                        Zoom = .25;
                        this.AutoScrollMinSize = new Size ((int)(m_Bitmap.Width
* Zoom), (int)(m_Bitmap.Height * Zoom));
```

```
            this.Invalidate();
        }
        private void OnZoom50(object sender, System.EventArgs e)
        {
            Zoom = .5;
            this.AutoScrollMinSize = new Size ((int)(m_Bitmap.Width
* Zoom), (int)(m_Bitmap.Height * Zoom));
            this.Invalidate();
        }
        private void OnZoom100(object sender, System.EventArgs e)
        {
            Zoom = 1.0;
            this.AutoScrollMinSize = new Size ((int)(m_Bitmap.Width
* Zoom), (int)(m_Bitmap.Height * Zoom));
            this.Invalidate();
        }
        private void OnZoom150(object sender, System.EventArgs e)
        {
            Zoom = 1.5;
            this.AutoScrollMinSize = new Size ((int)(m_Bitmap.Width
* Zoom), (int)(m_Bitmap.Height * Zoom));
            this.Invalidate();
        }
        private void OnZoom200(object sender, System.EventArgs e)
        {
            Zoom = 2.0;
            this.AutoScrollMinSize = new Size ((int)(m_Bitmap.Width
* Zoom), (int)(m_Bitmap.Height * Zoom));
            this.Invalidate();
        }
        private void OnZoom300(object sender, System.EventArgs e)
        {
```

```
                Zoom = 3.0;
                this.AutoScrollMinSize = new Size ((int)(m_Bitmap.Width
* Zoom), (int)(m_Bitmap.Height * Zoom));
                this.Invalidate();
        }
        private void OnZoom500(object sender, System.EventArgs e)
        {
                Zoom = 5;
                this.AutoScrollMinSize = new Size ((int)(m_Bitmap.Width
* Zoom), (int)(m_Bitmap.Height * Zoom));
                this.Invalidate();
        }
```

In the filter options many function are included. All these functions take input as bytes. For these the image is converted to bytes using bitmap filter .It is done by the following

```
BitmapDatabmData=b.LockBits(new   Rectangle(0,0,b.Width,b.Height),
                                ImageLockMode.ReadWrite,
PixelFormat.Format24bppRgb);
```

The gray scaling function is defined as follows

Filter_GrayScale(object sender, System.EventArgs e) is activated when the gray scale button is pressed(refer appendix 2). In this it call the function

boolGrayScale(Bitmap b)

This function changes the red, green and blue components as

*p[0] = p[1] = p[2] = (byte)(.299 * red + .587 * green + .114 * blue);*

where p[0] is for red, p[1] for green and p[2] for blue.

Filter_Brightness(object sender, System.EventArgs e) function call the function to make changes in brightness.

Brightness(Bitmap b, intnBrightness)

It is implemented as follows

```
for(int y=0;y<b.Height;++y)
```

```
{
    for(int x=0; x < nWidth; ++x )
            {
            nVal = (int) (p[0] + nBrightness);
            if (nVal < 0) nVal = 0;
            if (nVal > 255) nVal = 255;
            p[0] = (byte)nVal;
             ++p;
            }
            p += nOffset;
            }
```

Here b is the image in bytes and nBrightness is the limit for brightness value.The pixel values are adjusted according to the specified value(refer appendix 3).

Filter_Contrast(object sender, System.EventArgs e) calls the function to change contrast of the image. The contrast function is similar to the brightness function and do the same operation except that it is done to the red, green and blue components of the image

Contrast(Bitmap b, sbytenContrast)

Here b is the input in bytes and nContrast is limiting value.

It is implemented as follows

```
for(int y=0;y<b.Height;++y)
  {
    for(int x=0; x < b.Width; ++x )
     {
        blue = p[0];
        green = p[1];
        red = p[2];
    pixel = red/255.0;
            pixel -= 0.5;
        pixel *= contrast;
```

```
                pixel += 0.5;
                if (pixel < 0) pixel = 0;
                if (pixel > 255) pixel = 255;
                p[2] = (byte) pixel;
                pixel = green/255.0;
                pixel -= 0.5;
                pixel *= contrast;
                pixel += 0.5;
                pixel *= 255;
                if (pixel < 0) pixel = 0;
                if (pixel > 255) pixel = 255;
                p[1] = (byte) pixel;
                pixel = blue/255.0;
                pixel -= 0.5;
                pixel *= contrast;
                pixel += 0.5;
                pixel *= 255;
                if (pixel < 0) pixel = 0;
                if (pixel > 255) pixel = 255;
                p[0] = (byte) pixel;
                p += 3;
                }
                p += nOffset;
        }
```

bool Gamma(Bitmap b, double red, double green, double blue) is the gamma function which take image in bytes as input and also take parameters for red, green and blue components in image. The gamma function make changes in gamma values of red, green and blue components as specified. It is defined as

*redGamma[i] = (byte)Math.Min(255, (int)((255.0 * Math.Pow(i/255.0, 1.0/red)) + 0.5));*

*greenGamma[i]=(byte)Math.Min(255, (int)((255.0 *Math.Pow(i/255.0,1.0/green)) + 0.5));*

*blueGamma[i] = (byte)Math.Min(255, (int)((255.0 * Math.Pow(i/255.0, 1.0/blue)) + 0.5));*

The red, green and blue color of image can be varied using color function. It is defined as*boolColor(Bitmap b, int red, int green, int blue)*. In this function the red, green and blue components are changed as

```
nPixel = p[2] + red;
nPixel = Math.Max(nPixel, 0);
p[2] = (byte)Math.Min(255, nPixel);
nPixel = p[1] + green;
nPixel = Math.Max(nPixel, 0);
p[1] = (byte)Math.Min(255, nPixel);
nPixel = p[0] + blue;
nPixel = Math.Max(nPixel, 0);
p[0] = (byte)Math.Min(255, nPixel);
```

The invert function is defined as follows*boolInvert(Bitmap b)*

It is just taking the opposite color of the current component. That is for example if the color component is 00,then the opposite we get is FF(255-0) (refer appendix 4).

It just add or subtract the component value from 255.It is implemented as

```
for(int y=0;y<b.Height;++y)
        {
                for(int x=0; x <nWidth; ++x )
                {
                        p[0] = (byte)(255-p[0]);
                        ++p;
                }
        p += nOffset;
        }
```

The watermarking (refer appendix 5) is implemented as follows

```
if (this.Image != null)
        {
this.tempImage = (Image)this.Image.Clone();

            Graphics g = Graphics.FromImage(tempImage);

            Color color = Color.FromArgb(this.TextOpacity,
colorPreviewPic.BackColor);

            Brush brush = new Pen(color).Brush;

g.DrawString(this.WatermarkText, this.TextFont, brush, this.TextPosition);

if (this.chkPreview.Checked)
            {
PreviewPic.Image = tempImage;
            }
```

The fontlist are loaded from the installed font collection in c# library as follows

```
cmbFontList.Items.Clear();
using (System.Drawing.Text.InstalledFontCollectionfontCollection = new
System.Drawing.Text.InstalledFontCollection())
        {
foreach (FontFamily family in fontCollection.Families)
            {
cmbFontList.Items.Add(family.Name);
            }
        }
```

Fontsize for watermarking text is selected as follows

```
cmbSizeList.Items.Clear();
for (int i = 8; i <= 72; i += 2)
        {
```

```
cmbSizeList.Items.Add(i);
        }
```
Color is given for watermarking text as
```
if (textColorDialog.ShowDialog() ==
System.Windows.Forms.DialogResult.OK)
        {
colorPreviewPic.BackColor = textColorDialog.Color;
txtColorValue.Text = textColorDialog.Color.ToString();
this.DoWatermark();
        }
```
The text position is adjusted by
```
get
        {
PointF p = new PointF(Convert.ToSingle(xUpDown.Value),
Convert.ToSingle(yUpDown.Value));
return p;
        }
```
The other features given for watermarking text are underline, bold, italic, strike off and is done as follows
```
get
        {
            _fontStyle = FontStyle.Regular;

if
(chkIsBold.Checked&&chkIsItalic.Checked&&chkIsStrikeout.Checked&&chkIs
Underline.Checked  )
    _fontStyle =
System.Drawing.FontStyle.BoldSystem.Drawing.FontStyle.Italic |
System.Drawing.FontStyle.Strikeout | System.Drawing.FontStyle.Underline;
            }
else
            {
```

```
if (chkIsBold.Checked)
                {
                    _fontStyle = _fontStyle | System.Drawing.FontStyle.Bold;
                }
if (chkIsItalic.Checked)
                {
                    _fontStyle = _fontStyle | System.Drawing.FontStyle.Italic;
                }
if (chkIsUnderline.Checked)
                {
                    _fontStyle = _fontStyle | System.Drawing.FontStyle.Underline;
                }
if (chkIsStrikeout.Checked)
                {
                    _fontStyle = _fontStyle | System.Drawing.FontStyle.Strikeout;
                }
            }
return this._fontStyle;
        }
```

Another feature given is the opacity

```
get
        {
return ((256 / 100) * opacityTrack.Value);
        }
set
        {
opacityTrack.Value = value;
        }
```

These processed images are input to the next stage, i.e. for ordering and from that our system will generate a video sequence.

In the next stage we are proposing to do a partial temporal ordering for these images. After ordering it is given to the video maker and produce the animated scenes.

IMAGE SORTING

For the purpose of image sorting we can use any of the sorting methods like bubble sort, insertion sort, selection sort etc. Here we use a simple sorting mechanism.

At first we make a comparison between two images as follows

```
string img1_ref, img2_ref;
    img1 = new Bitmap(fname1);
    img2 = new Bitmap(fname2);
if (img1.Width == img2.Width && img1.Height == img2.Height)
    {
for (int i = 0; i < img1.Width; i++)
        {
for (int j = 0; j < img1.Height; j++)
            {
                img1_ref = img1.GetPixel(i, j).ToString();
                img2_ref = img2.GetPixel(i, j).ToString();
if (img1_ref != img2_ref)
                {
count2++;
flag = false;
break;
                }
count1++;
            }
        }
if (flag == false)
MessageBox.Show("Sorry, Images are not same , " + count2 + " wrong pixels
found");
else
MessageBox.Show(" Images are same , " + count1 + " same pixels found and
" + count2 + " wrong pixels found");
    }
else
MessageBox.Show("can not compare this images");
this.Dispose();
}
```

The next phase is the image sorting. We can use any of the sorting techniques for this purpose. Here we used the simple selection sort for this purpose.

Selection sort algorithm:

Step 1: Start
Step 2: Read the array elements and length of array
Step 3: Repeat steps 4 for i=0 to i=length of array
Step 4: Repeat steps 5 for j=i+1 to j=length of array
Step 5: if(a[i]>a[j])
temp=a[i]
a[i]=a[j]
a[j]=temp
Step 6: Stop

The complete algorithm for comparison and sorting is as follows

Step 1: Start
Step 2: Read the first image as first element of image array and then read the remaining images into the array
Step 3:count<-0
Step 4: Repeat step 5 through 6 for i=0 to i=array length
Step 5: Var1<-count
Step 6: Repeat step 7 through 9 for j=i+1 to j=array length
Step 7: Compare a[i] and a[j]
Step 8:Read the pixel difference to count
Step 9:If(count < var1)
Swap a[i+1] and a[j]
Step 10: Stop

First image in the series is selected by the user. It is done at the time of reading the images. When the images in the sequence are read, the details of these images are saved into a database. When we select an image as the first image its status field in the database is set as ‘Y’ and all other images’ status as ‘N’. So while reading the images into an image array the one with status field ‘Y’ in the database is read as first element and then the remaining images in the folder.

Then in the comparison part, set var1=0 and count=0 .The first image is compared first with the second image and the pixel difference is stored into a variable count. Then compare var1 and count . Set var1=count . Then first image is compared with the third image and pixel difference is stored to a variable count . If var1>count , then the second and third images will be swapped. Otherwise it is left unchanged. Like this the first image is compared with all other images in the array and according to the pixel difference they are rearranged. Then the second image is fixed and it is compared with all remaining images in the array for fixing the third according to the pixel difference. Like this the sorting process progress and finally we get the sorted image array.

This images from image array is save as it is in the array to a folder. The video making part is applied to this folder to generate the video sequence.

Then a novel image median filtering algorithm based on incomplete quick sort algorithm is proposed to improve the filtering speed. The new algorithm considers in detail the characteristic of image median filtering (In median filtering algorithm, the sorting operation of all pixel values is not necessary, the median value can be given by many other methods), and can give the median value by only sorting part of the pixels value in the neighborhood, thus it can reduce many data move operations, and then greatly improve the speed of image median filtering. Algorithm analysis and a

lot of experiment results show that, the new algorithm greatly improves the speed of image median filtering, and can keep the edge, outline, texture and much other information to a great extent.

We can use any of the sorting techniques. Since it is sorting images, some techniques offer low accuracy only.

VIDEO MAKING

The video making part is done using the in-built components in c#. Many library classes are there in c# for the purpose. So it is more easy to implement this part. The implementation is done as follows

```
using System;

usingSystem.Diagnostics;
usingBytescoutImageToVideoLib;

namespaceSimpleSlideshow
{

class Program
{
staticvoid Main(string[] args)

{
Console.Write("Converting JPG slides into video, please wait..");

// CreateBytescoutImageToVideoLib.ImageToVideo object instance

ImageToVideo converter = newImageToVideo();
```

```
// Activate the component
converter.RegistrationName = "demo";

converter.RegistrationKey = "demo";

// Add images and set the duration for every slide
Slide slide;

slide = (Slide) converter.AddImageFromFileName("..\\..\\..\\..\\slide1.jpg");

slide.Duration = 3000; // 3000ms = 3s
slide = (Slide) converter.AddImageFromFileName("..\\..\\..\\..\\slide2.jpg");

slide.Duration = 3000;
slide = (Slide) converter.AddImageFromFileName("..\\..\\..\\..\\slide3.jpg");

slide.Duration = 3000;

// Set output video size
converter.OutputWidth = 400;

converter.OutputHeight = 300;

// Set output video file name
converter.OutputVideoFileName = "result.wmv";

// Run the conversion
converter.RunAndWait();

Console.WriteLine("Conversion is done. Press any key to continue..");

Console.ReadKey();
```

```
// Open the result video file in default media player
Process.Start("result.wmv");

}
}
```

The video stream is generated using the AVIFile functions. The most important AVIFile functions into three easy to use C# classes that can handle the following tasks:

Read images from the video stream.

Decompress a compressed video stream.

Compress an uncompressed video stream.

Change the compression of a video stream.

Export the video stream into a separate .avi file.

Export the audio stream into a .wav file.

Copy a couple of seconds from audio and video stream into a new .avi file.

Add sound from a .wav file to the video.

Create a new video stream from a list of bitmaps.

Add frames to an existing video stream, compressed or not.

Insert frames into a stream.

Copy or delete frames from a stream.

These features cover the common use cases like creating a video from a couple of images and a wave sound, extracting the sound track from a video, cutting out short clips, or grabbing a single picture from a movie.

AVIManger manages the streams in an AVI file. The constructor takes the name of the file and opens it. Close closes all opened streams and the file itself. You can add new streams with AddVideoStream and AddAudioStream. New video streams are empty, Wave streams can only be created from Wave files. After you have created an empty video stream, use the methods of VideoStream to fill it.

Create a video stream

There are two methods for creating a new video stream: create from a sample bitmap, or create from explicit format information. Both methods do the same, they pass their parameter on to VideoStream and add the new stream to the internal list of opened streams, to close them before closing the file.

The video generating part is implemented with the help of BytescoutImageToVideo Library. BytescoutImageToVideo library has inbuilt functions in it to produce a video from a series of images. We make use of it to generate an animated output. The slide duration is specified explicitly to make it. It is implemented as a simple slide show with the slide duration is adjusted to produce it as a video sequence.

7.CONCLUSION

This project has been developed as versatile and user-friendly. And the design of the system is in such a good manner that supports implementation and software maintenance in a proper way.

In this project, we describe a system that allows the user to quickly and easily produce a compelling-looking animation from a small collection of high resolution stills.. Using our system, an animated scene can be generated in minutes. We show results for a variety of scenes.

We have tried our level best to implement the features as possible with the available resources in this confined time for the project.