The Chinese University of Hong Kong
Department of Computer Science and Engineering
Final Year Project Report
2nd semester, 2003-2004

Video Object Tracking and Replacement for Post TV Production

By
FYP Group LYU0303

Group Members:
Tong Pak Hin, Tim 01600513
Yuen Pak Kei, Ivan 01578001

Index

Chapter 1

Introduction

1.1 About Post-TV Production and Video Processing

In the time of 1960's to 1970's, color televisions have become lower in price and more available to the general public [1]. At the same time, watching TV programmes became an important part of people's life as an entertainment. Since more and more people were watching television programmes, television broadcasting companies were willing to produce more and more television programs. This was because with more audience sticking to their television programmes, more people would ask the television broadcasting companies to advertise their own commodities. Income of TV broadcasting companies comes mainly from advertising, anyway. This is because most of the television programmes are broadcast free of charge.

As there are more and more TV programmes available, less people are willing to pay for a movie ticket nowadays. To ensure stable income, movie producers like to include some special effects in their products such that people think watching a movie is worth the money. Back to the television broadcasters, since there are too many television programmes available, television broadcasting companies also want to give their television programmes something surprising in order to attract the audience.

The special effects are, however, often dangerous in nature. For example, if we want to produce a "flying man" effect, we must wind several thin wires around the actors and then drag him up to the sky using a helicopter, a crane or something like that. What will happen if the wire breaks when the actor is just in the mid-air? No one can predict the consequence. Also, the thin wires can still be seen by the audience. This will greatly reduce the reality feeling of the effect.

Sometimes people will try to replace the real scene by a fake effect when producing a video. Take the example above: If we generate a "flying man" effect by replacing everything (except the actor) with a fake background, the flight of our "flying man" may be unnatural. This is because the actual moving speed of the man and the "sky" background may not match. There is actually a dilemma between using real effects and using simulated effects: Safety and reality.

According to this reason, some suggestions have been given to solve the dilemma. One of them is, produce two real video clips and merge the necessary parts together. Since the special effects can be seen only when the video is being processed but not during the recording phase, this kind effect producing technique is called "Post-TV production" or simply "video processing".

1.2 Object Tracking and Replacement

As computers have become faster, they are more involved in complex calculations and manipulations as well. So, people tried to think of using a computer to merge two relevant video clips – by just extracting the needed part from both clips and discard all other things.

A typical example of movie that uses plenty of special effects is *Titanic*, directed by James Cameron. *Titanic*, the Oscar Award winner of Best Visual Effect in 1997, [2] has adopted various computer generated effects throughout the entire video. This is why the film seems so realistic. Even the ship itself is just a special effect (of course it is not possible to build a new ship just for making a movie).

The use of computer has reduced the danger for actors, but it does not reduce the work for people who process the video afterwards. People still have to work long for drawing very precise effects using computer. This is why movies with a lot of special effects have very high production cost. In the case of *Titanic* and *Terminator 3* the production costs are 137 million and 175 million U.S. dollars respectively. [3]

At this point, people maybe think whether computers can perform the processing such that no human intervention is needed. This can be done if the computer can find out what exactly it is going to modify in the video clips. Usually, the things to be processed can be treated individually as "video objects". If the computer can find out the video objects in the video, then it can almost perform any process automatically.

The step of finding out where the video objects are is called "object tracking".

If the above "flying man" sample is being produced in this way, it can be as the following: Produce two video clips. One with the actor hung 1 foot above the ground and another with a big doll hung in the mid-air. By replacing the doll with the image of the actor, it will then look like a real actor is in the mid-air.

In a brief summary, video object tracking has the following advantages:
- Safe: Actors do not have to perform dangerous actions.
- Cost saving: Less human intervention. One video clip can produce different effects.
- Producing fancy effects those are not possible in reality (e.g. human flying).

Figure 1.2.1 Screenshots of *Titanic* and *Terminator 3 – Rise of Machines*, two films using extensive computer generated effects.

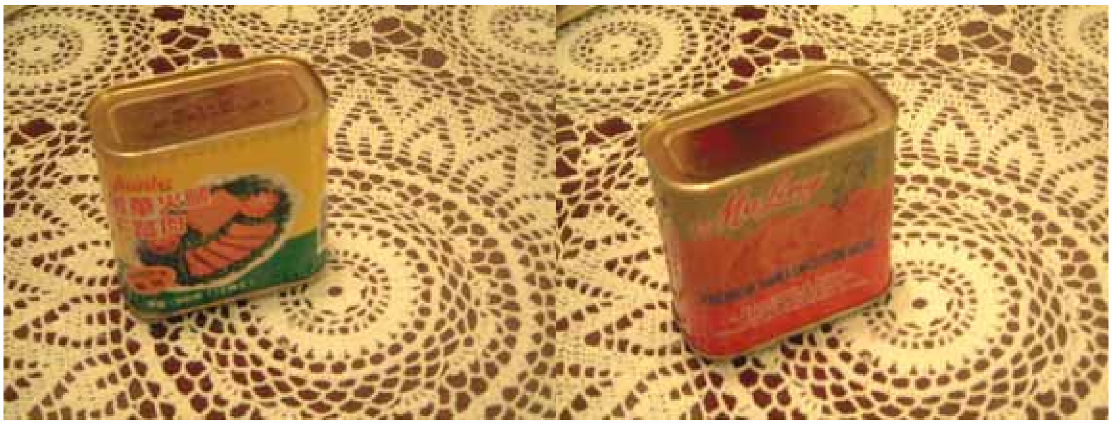


Figure 1.2.2 Two cans with the same shape but different brand names. Can we take only one photo but will produce two effects in the future?

1.3 Objective of Developing the Project

As video object tracking and replacement system has so many advantages, it is worthy that such system can be developed. Since video processing is still a complex process in comparison to other computer processes, we will try to work on a simplified version of the system. The system will at least:

- Detect and locate simple video objects (e.g. cubes, cylinders and cones).
- Find out the orientation and current state of the objects (e.g. whether there is a shadow or anything those shade part of the object away from the video).
- Replace the surface of the video object by a predefined texture, while keeping any other constraints (e.g. if part of the original video object is shaded away, then after the process that part will still be shaded away).

Chapter 2

Overview of the Project's Working Principle

2.1 Difference between Humans and Computers in Object Recognition

In the last chapter it has been mentioned that a video tracking and replacement system can:
- Detect and locate simple video objects.
- Find out the orientation and current state of the objects.
- Replace the surface of the video object by a predefined texture, while keeping any other constraints.

The basic part of the system is the first two points. Although the last point is complex, it still replies on the first two points to be done. So in Project Part One we will focus on the first two points.

Locating an object may sound easy to us human beings. But, in fact it is not easy for the computers to perform. This is because human brains and computers work in totally different ways.

Humans read in an image by looking at the whole picture. Although humans cannot memorize the exact characteristics of the things in it, they do get a brief image of the whole picture in a very short time. Computers, on the other hand, read in image files byte-by-byte. Although a computer won't miss anything (in fact pixels) in an image, it cannot get any information of the whole picture by just getting a 2D-array of the image pixels. Extra work can be done to get some of the information, but it will cost too much time.

Humans are very good at memorizing images. Although not as accurate as a computer does, humans are much faster in memorizing and very little information is needed. Thus, human has enough prior knowledge to point out almost everything during recognition. Computers can recognize object if and only if detail information is given. For example, if we need a computer to recognize a quadrilateral, you may need to give it a lot of equations.

The biggest problem is that a computer can only interpret in a definite way. That means no errors or missing information is allowed. When we shade a corner away from a cube, we know that the surface is a square since we know that cubes do not have pentagon surfaces. However, computer determines a cube by using its surfaces instead. When a corner is shaded, it fails to recognize the quadrilateral surface and hence fails to detect the cube. In one sentence, computer works without fuzzy logic.

Take an example. Suppose both human and computer look at the following figure.

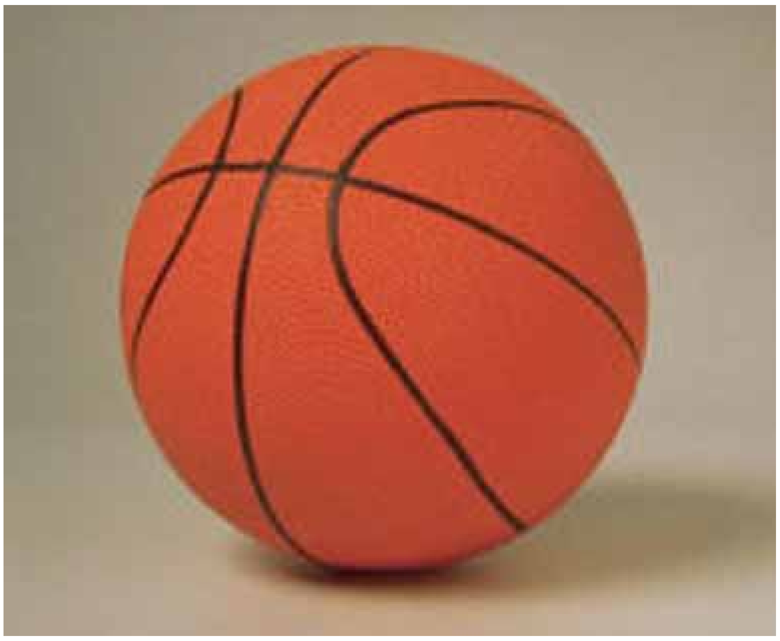


Figure 2.1.1 Try to guess what it is!

When a human looks at this figure, his attention will be attracted by the bright orange color. He looks and follows the edge and knows that this is a sphere. By looking at the black line pattern and from his previous knowledge, he knows that the object should be a basketball.

For a computer it’s a totally different story. The computer scans the image file row by row, column by column (no matter what format the image is currently, maybe we need a new image format). Then it points out which the major object should be by counting the number of pixels of a certain color. Next, it roughly observes the trend of edge points to estimate that it is a circle (cannot derive any 3D information by merely looking at the color). Up to this point, the computer can only conclude that there is an “orange circle with black pixels in it”, which is far away from the truth.

2.2 Applying the Human Principle of Image Recognitions onto Computers

Although a computer does not work in the same way as a human being, we can give it some instructions so that it will try to compute an image in a more “human” way.

It has been mentioned that main characteristics of an object is its brightness, color, size and shape. So, we will concentrate our works in these particular areas.

To detect a pixel of a particular color and brightness, we have to make use of the current image file format. That is, a simple image file reader will be implemented to meet the requirement.

Among these factors, the last one is difficult to be determined. In order to determine the existence of a particular shape, we will derive more information out of the edges in addition to the direction where they are going to extend to.

After performing the above tasks, it should be able to figure out a particular object in an image. As a video consists of a sequence of still images which may look very similar to each other, we can apply some algorithms such that previous information obtained can be reused in the latter processes. The time saved can be significant since even a short video of several seconds may contain over 1,000 video frames already.

Although there are some similar products those can perform similar video processing functions (e.g. ARToolKits), they work only with very simple video objects like square. And, they do not allow occultation (the object must not be covered). Finally, they are still too inefficient in comparison to current video need (320x240 at 15fps, not even reach the quality of a typical VCD) [4].

Summarizing the above points, the entire project can be roughly divided into the following parts:

-Simple bitmap reader/writer
-RGB/HSV converter
-Edge detector
-Edge equation finder
-Equation processor
-Translation detector
-Texture mapper (which will be included in Part 2 of the report)

In the following chapters, the functions of the individual parts will be discussed in detail.

Chapter 3

Entering the World of Digital Image Processing

3.1 Binary File Reading and a Simple Bitmap Reader

Although we are trying to apply human image recognition techniques on a computer system, it is inevitable that we are placed under the restriction of current computer technology. Up to now, at least we can't ask the computer not to read a file byte-by-byte but looking at the whole image globally! So, we need to find out which format will suit our need the most.

There are different kinds of image file format available around us. In order to compare the characteristics of different file formats, let us look at the following table first:

File Formats	Bitmap (*.bmp)	Graphic Interchange Format (*.gif)	Joint Photographic Experts Group (*.jpg)
File Size (byte)	1,400,054	161,655	44,883
Color Depth (bit)	24	8	24
Data Lost	No (Raw)	No (If input is 256 colors)	Yes

Figure 3.1.1 Comparison of three different image file formats. The file is 800x600 in dimension. [5]

From the table we have seen that uncompressed bitmap files (*.bmp without RLE, Run Length Encoding) have largest file size since it is uncompressed. However, video processing does need a high speed computation, and using bitmap files can reduce the extra compression/decompression time. For this reason, we will try to work with bitmaps at this stage. In a later time, we will build an improved version of the bitmap files. That is, the file will not store the pixels from left to right, down to up sequence any more. The pixels will be stored in a way such that the computer will obtain most of the color information by just reading the first portion of the bitmap file.

Although the sizes of bitmap files are large, they can still be handled by most modern computer systems:

After the bitmap file has been read into the main memory, it can be further processed to make any changes.

3.2 Color Representation Model

We have seen that there are various kinds of bitmap file formats. In fact, there are different ways of representing a color also. For example, a typical CRT display monitor will interpret 3 signals (Red, Green and Blue, known as the RGB color representation) and adjust the strength of the electron guns. Color printers, on the other hand, use 4 signals (Cyan, Magenta, Yellow and blacK, known as the CMYK color representation) and spray the different ink onto the paper.

Human eyes are more sensitive to the change of brightness than to the change of color. The JPEG file format uses a lossy compression that discards color change information those cannot be detected by human eyes. [6] In order to make the representation of color human-friendly, people has introduced a new color model called HSV (Hue, Saturation and Value. Value is the same as brightness). And, this will be the main color model we will use throughout the project.

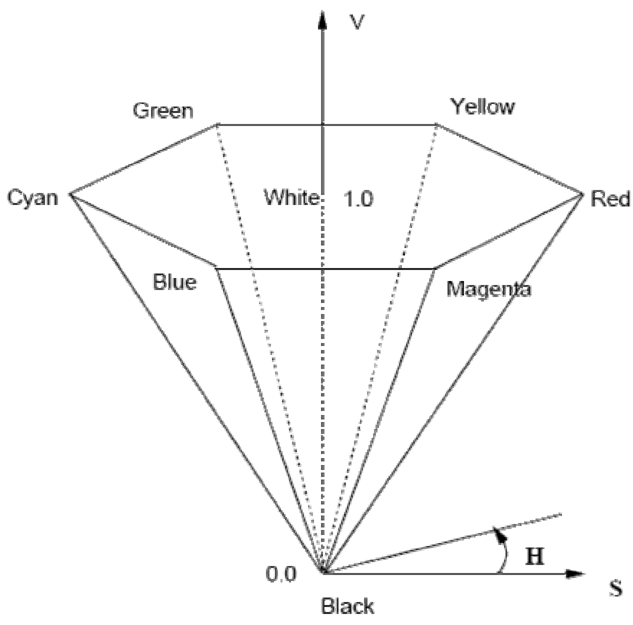


Figure 3.2.1 The HSV hexcone model [7]

Hue is the color component. It is a 360 degree system that records the color information. For example, 0 degree represents red, 120 degree represents green and 240 degree represents blue.

Saturation is the "concentration" of a color. Take a very easy example, if we add milk to a cup of black coffee, it will turn dark brown. If the milk is in excess, the whole cup of coffee may become pale brown. When adding milk to the cup of coffee, we are in fact decreasing the saturation value of the cup of coffee since the "color concentration" of it has been decreased.

Brightness value is the part that stimulates our vision most. This is because humans have far more rod cells (which detects brightness) than cone cells (which distinguishes color). When the brightness of a color increases, the ability to draw attention will also increase. If we illuminate an area we are increasing its brightness. Oppositely if we shade an area, we are decreasing its brightness.

HSV scheme works well when we need human-like color recognition. [8]

3.3 The RGB/HSV Color Converter

From the previous page we know that for the convenience of computer hardwares (not humans), bitmap files stores the color information in RGB format instead of HSV format. In order to make the algorithms work on a computer, we must first convert the color information format from RGB to HSV first. After the process, convert HSV information back to RGB and store it back to a magnetic disk or whatever else. So, an RGB/HSV converter has been integrated into the system for this purpose.

The RGB/HSV converter uses the following formulae for the color model conversion.

$$
\begin{aligned}
S &= \begin{cases} \frac{\max(R,G,B)-\min(R,G,B)}{\max(R,G,B)}, & \text{if } \max(R,G,B) \neq 0 \\ 0, & \text{if } \max(R,G,B) = 0 \end{cases} \\
H &= \begin{cases} \frac{(-B+G)\pi/3}{\max(R,G,B)-\min(R,G,B)}, & \text{if } R = \max(R,G,B) \\ \frac{(-R+B)\pi/3}{\max(R,G,B)-\min(R,G,B)}, & \text{if } G = \max(R,G,B) \\ \frac{(-G+R)\pi/3}{\max(R,G,B)-\min(R,G,B)}, & \text{if } B = \max(R,G,B) \\ \text{undefined}, & \text{if } R = G = B \end{cases} \\
V &= \max(R,G,B)
\end{aligned}
$$

$$
\begin{aligned}
I &= \text{floor}\left(\frac{3H}{\pi}\right) \\
f &= H - I \\
p &= V(1-S) \\
q &= V(1-Sf) \\
t &= V(1-S(1-f))
\end{aligned}
$$

$$
[R\ G\ B] = \begin{cases} [V\ t\ p], & \text{if } I = 0 \\ [q\ V\ p], & \text{if } I = 1 \\ [p\ V\ t], & \text{if } I = 2 \\ [p\ q\ V], & \text{if } I = 3 \\ [t\ p\ V], & \text{if } I = 4 \\ [V\ p\ q], & \text{if } I = 5 \\ [0\ 0\ 0], & \text{if } S = 0 \end{cases}
$$

Figure 3.3.1 RGB/HSV conversion formulae [7]

There is one point to be mentioned when using the above formulae. When a color is on a grayscale (i.e. R=G=B), the hue (color) value is undefined. For simplicity, we just store the value of hue as 0 in this case. In image processing stage, a program must take care of the saturation ("amount"of color present) and brightness values first. Otherwise the color may get mixed up with red, which also has hue value 0.

Originally hue is noted in a 360 degree system. In order to fit it into a byte system, it is transformed into a value such that its maximum is at 255. Then, the HSV color model can be stored by 3 bytes just as the RGB color model.

Chapter 4

Knowing More from the Image Files

4.1 Simple Edge Detector

We often distinguish two different objects by telling the differences between them. And, this principle can be applied onto a computer system as well. A computer can detect an object if it is significantly different from the surroundings. Usually, the difference is caused by a change in color. [9]

Assume we are going to find out an object of a particular color (suppose we know that exactly, or that value is being estimated by a computer imaging system), then when the computer encounters a color change to our desired color, we know that the object is being reached. This is true given that nothing else in the picture has the same color. In the case of leaving a particular pixel of desired color into a pixel of different color indicates that the scanning has left the target object.

In the following figure, suppose we are going to detect an object which is pure red in color:

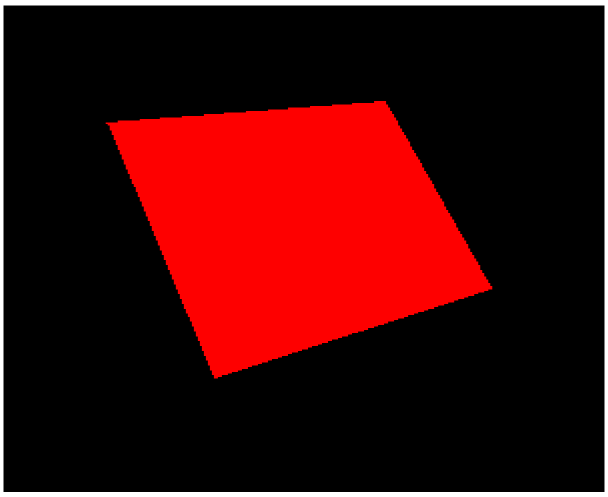


Figure 4.1.1 A red polygon

Since a computer will scan through the image from left to right, it will encounter a change in color when it reaches the polygon. In this case, the color changes from black (0, 0, 0) to red (0, 255, 255).

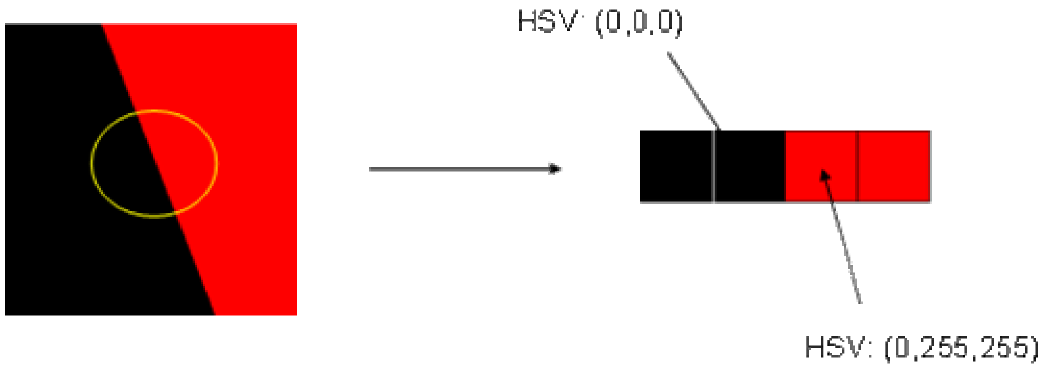


Figure 4.1.2 Color change at the boundary

As we have set red as the color of our target image object, the computer knows that it has reached the boundary. It will mark that point as an "edge point", being recorded in a 2D-array. Similarly, the computer will mark an edge point in the 2D-array again when it is leaving the target image object.

If we draw the edge points in a separate picture, the processed image may look like:

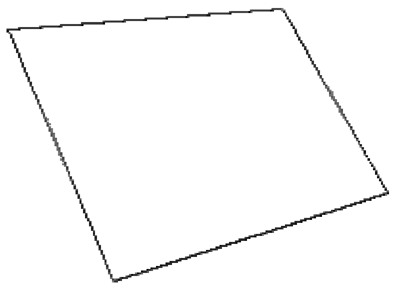


Figure 4.1.3 "A red polygon" being processed by the simple edge detector

Since the picture is being drawn by hand, the edges are very sharp and are detected fairly easily by the computer. If we use real boxes, we may encounter problems such as noises.

Below is a photograph of a real box. We can see that if it is being processed by our software, we may get some "extra points" which may affect future processing of the image:

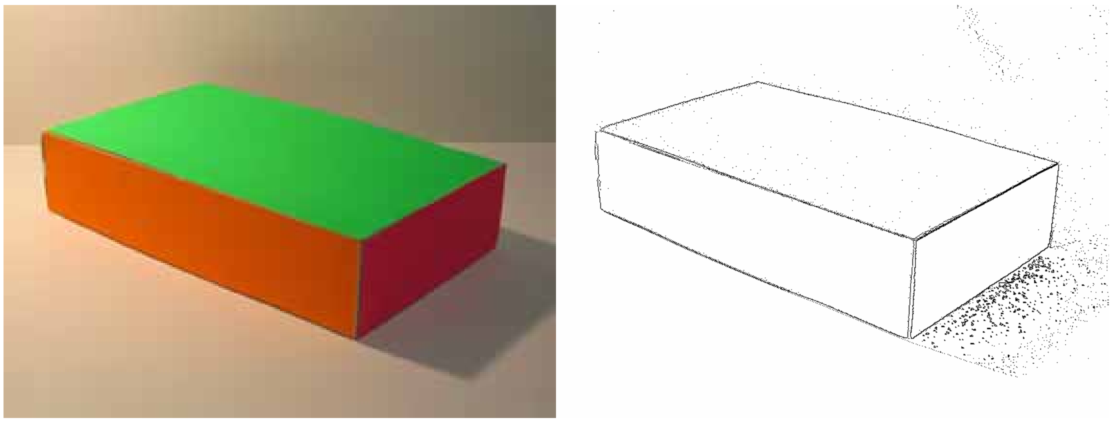


Figure 4.1.4 An image before and after edge detection. Note the existence of noise points due to reflection and ambient lighting.

To solve the problem we have two solutions. One is to take photo in a controlled environment such that reflection is minimized and the light source is white in color. This ensures that other colors will not appear too often in the image.

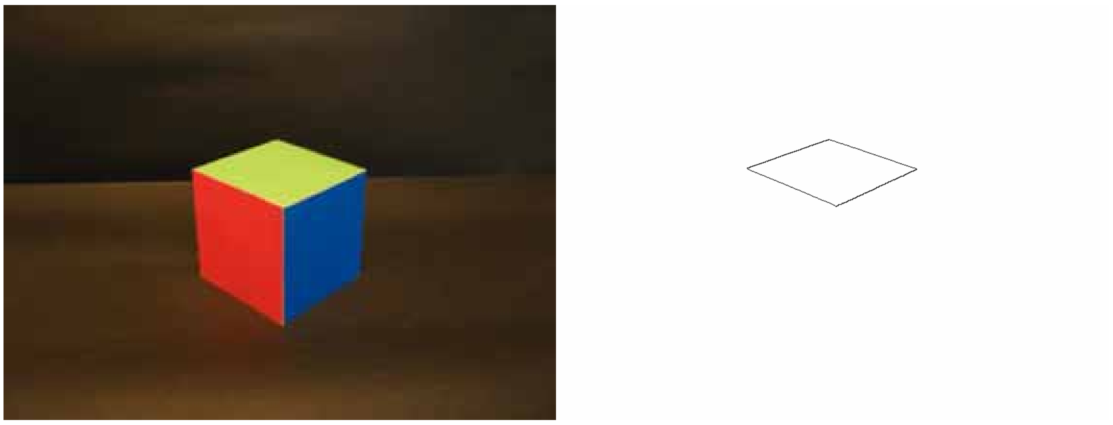


Figure 4.1.5 A box in a black background under a white light source. For simplicity we just detect the green surface.

Another method is to apply smoothing before finding the edge points. Gaussian smoothing will eliminate most of the points which are "odd out" from the surrounding. And this will be helpful when a controlled environment is not available.

There are various kinds of smoothing method such as Gaussian smoothing [10], mean filter smoothing [11] and median filter smoothing [12]. Among these methods, the median method removes pepper salt noise points efficiently and is easy to compute so it is being applied in the project. We can see the effect of median filter smoothing in the following figures.

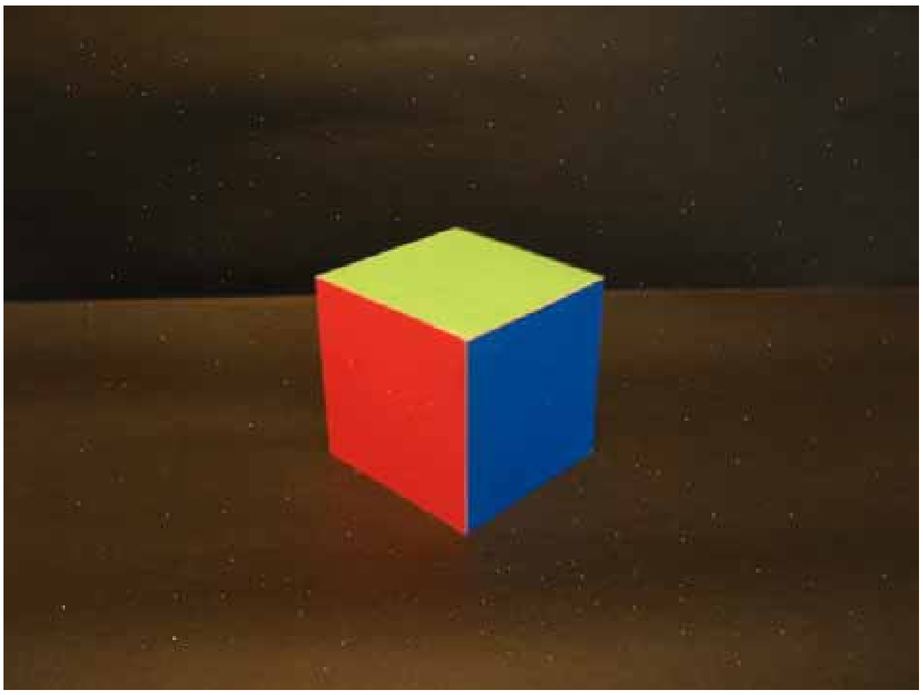


Figure 4.1.6 Before smoothing, the image is full of white dots which disturb edge detection.

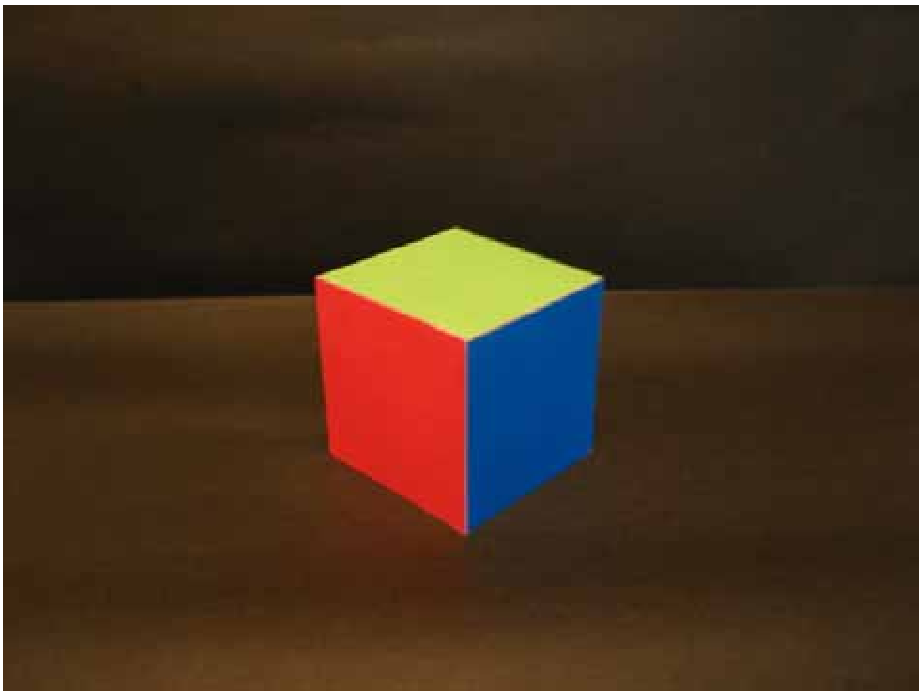


Figure 4.1.7 After smoothing, the image looks almost the same as the original one.

If noise points still exist, we can apply median filter smoothing multiple times, since median filter has little influence on the original image. The drawback is median filter smoothing is a bit time consuming since it involves sorting or color information.

At the same time, we can give some constraints such that the edge detector will only record the first entry point and the last exit point to make sure that no internal edges can be detected successfully. This will be useful to detect shadows or occultation in the future processes.

After edge detection, the edge points are exported as a “point-list”.

4.2 Equation Finder

Merely pointing out the edge points is meaningless. As every pixel in an image is being stored in a 2D-array, we can locate them by a Cartesian coordinate system. At the bottom right corner of an image, the pixel there is represented as (0, 0). This is also where the bitmap files start to record the pixels.

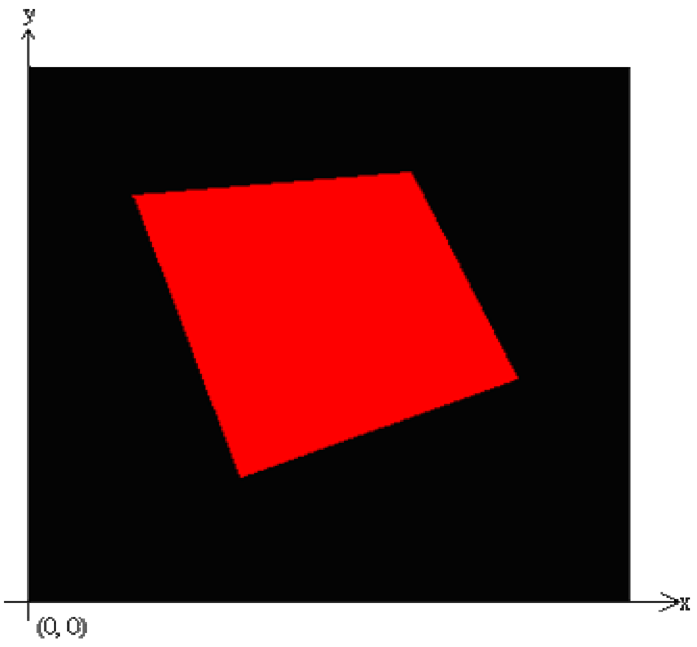


Figure 4.2.1 How the Cartesian coordinate system applies in the image format

Up to this stage, the computer just knows the existence of the target object, not its location. So, we must find out the exact relation between the edge points and the target object. So, we will try to determine whether the edge points can be fitted into several linear equations or not. By using inequalities, the exact location of the image object can be found.

An algorithm called Hough Transform can groups points those belong to the same linear equation and does further processing like line sharpening. [13] In the beginning, Hough Transform will raise an "election" between the edge points by testing which equation will pass through most of them. By successive trying and eliminating extra points, linear equations can be obtained.

Originally the "election" algorithm used by Hough Transform works with a polar coordinate system. To simplify the calculation process, we will let it work with the Cartesian coordinate system. As we are trying the equations of different angle (known slope) over a fixed edge point (known point), the output equations will be in point-slope form.

Suppose there are three edge points and one of them is called (x1, y1). The system will pass a linear equation through that point and mark down how many edge points has been passed through during the process. The test angles will be from 0 degree to 179 degrees. The following figure shows the election process when the trial angles are at 0 degree, 45 degrees, 90 degrees and 135 degrees respectively. Since the linear equation with angle 45 degrees has passed through the largest number of edge points, it is "elected" and the corresponding linear equation is stored in an equation list. Finally, all the points on that linear equation will be marked as scanned.

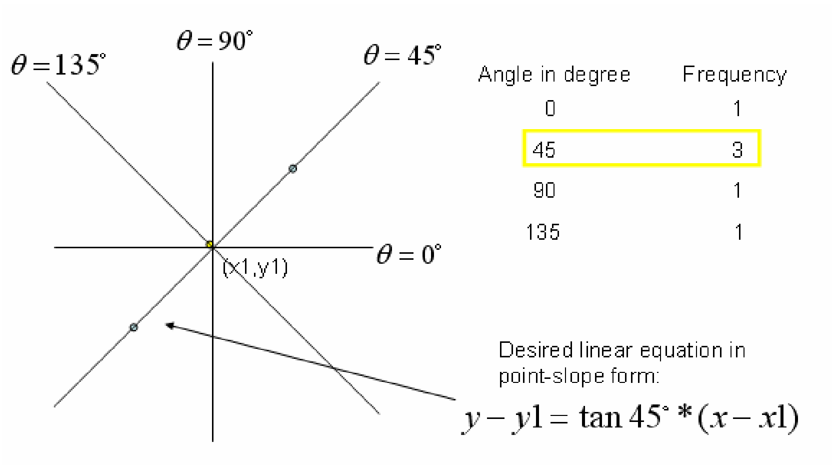


Figure 4.2.2 Cartesian equation electing algorithm

In real world, even the edges of a cube cannot be completely straight due to camera distortion or deformation of the cube. These may produce very short edges or bend edges. To get rid of the problem, the election system will not accept an equation which passes through too few edge points in order to eliminate unwanted equations. Also, since we are applying continuous linear equations on discrete pixel array, tolerance is needed to tackle bend edges. However, using a large tolerance value may generate equations those are slightly away from the position of the original edges.

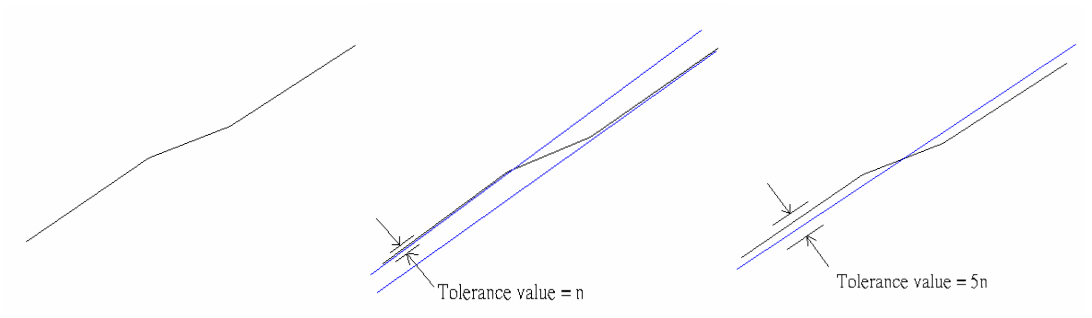


Figure 4.2.3 Equation finding using different tolerance values. Equations derived are shown in blue. For small tolerance value multiple equations may be derived. For large tolerance value there is only one equation derived. Note that the only equation does not stick to the original edge well.

The largest tolerance value is that an edge point can be 5 pixels away from the linear equation. The election procedure is repeated until all the edge points are marked or the number of edge points left is below a certain threshold.

We can see the effect of edge equation finder in the following figure.

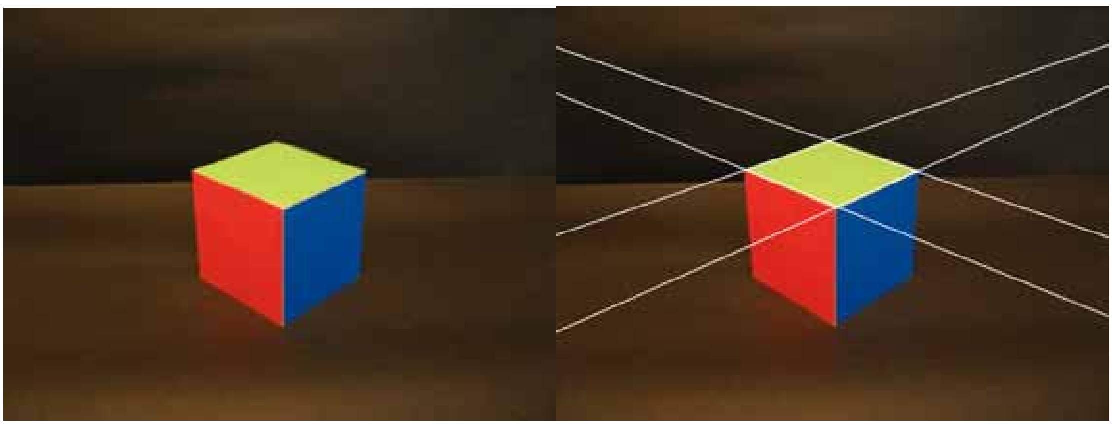


Figure 4.2.4 Edge equation finder. The green surface is being positioned by using the four linear edge equations.

Detection of curved surface will be included in a later stage.

4.3 Equation Processor

We have found out the exact location of the object, but it doesn't mean that the computer has all the information about the object. So, we may provide the computer with some prior knowledge such that the computer knows what do to next.

Suppose we are dealing with the surface of a cube. Obviously there should be four and only four equations to be derived from the edge points. It is not possible to have less than four equations or that means the surface cannot be seen. If the number of equations is more than 5, it may mean part of the surface is being covered.

From the equations we have obtained from the previous step, we may be able to deal with these problems. Suppose there are internal edge points those cannot be removed in the edge detection stage, then there will be internal linear equations which will cut the object into two halves.

We know that there should not be an equation which contains a point between any other two equations. Thus, we can use this information to remove possible occultation or shadow that covers part of the target object.

We can also try to solve the equations such that we can get the coordinates of the possible corner points. Corner coordinates are important data to future translation detection and texture mapping.

Chapter

5

Performance of Edge Detector

5.1 Processing Speed

The processing speed will be different if different images and different video objects are involved.

Usually, smaller images files with little noise points will be processed faster. The processing speed will also increase if fewer colors are involved in the image.

Take the previous color box image as an example, suppose we are going to find out the equations those surrounds the green surface.

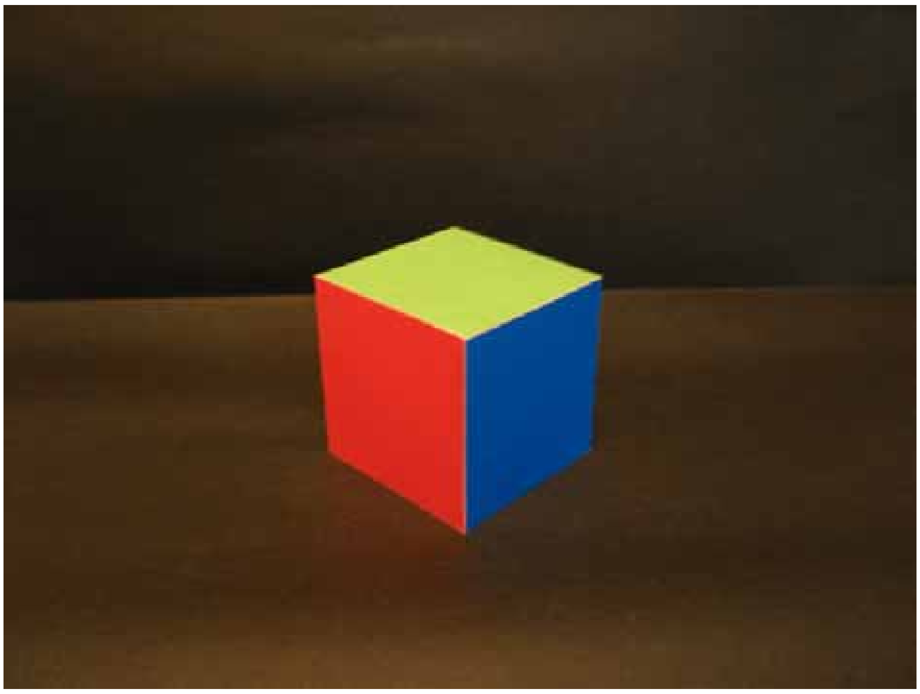


Figure 6.1.1 A sample image

Time (s)	352x288	640x480	800x600	1024x768
No smoothing	2.54	6.30	10.04	18.56
1st level smoothing	3.27	8.21	14.73	25.82
2nd level smoothing	3.61	10.86	16.23	28.52
3rd level smoothing	4.09	11.90	18.78	33.40

Figure 6.1.2 Process time required for different size and times of smoothing.

The video part is still in development stage so the process time will not be included.

5.2 Accuracy

Sometimes the image in a video may be blurred. So, we need to know how "blurred" the system can withstand.

Again, we use photograph of the same box.

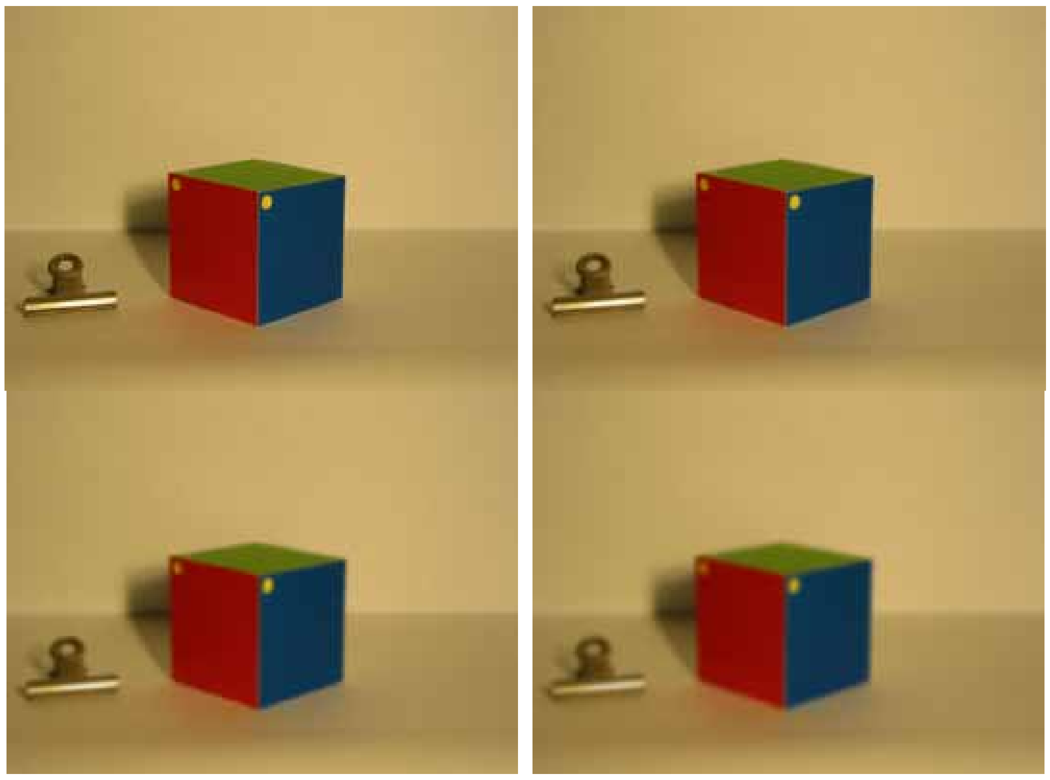


Figure 6.2.1 4 pictures of different blur level. Gaussian blurring is applied on the photos for radii of 0 (no blurring), 1, 2 and 3 pixels respectively

We will apply median filter smoothing for 1 time (that's the level which is required to work on the original photo) and see what result will be obtained.

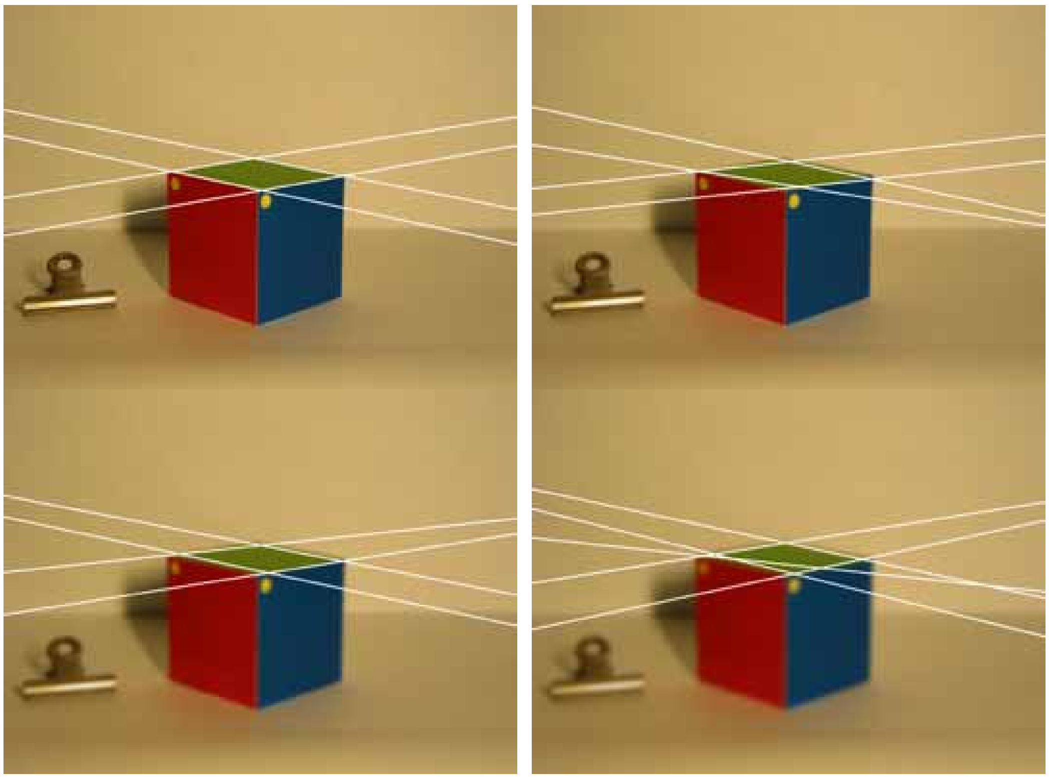


Figure 6.2.2 The result of edge equation finding.

We can see that for blurring level 1 and 2 the result is more or less acceptable. In the case of blurring level 3, there is an extra equation derived which may produce an error. In fact, a Gaussian blurring of radius 3 pixels is quite uncommon in real videos. The problem may be solved by adding marks to the surface.

5.3 Occultation Detecting

Sometimes the system has to deal with cases which part of the video object being covered up by another object. To simplify the problem, we assume that the two objects are different in color.

In fact, the algorithm does not detect occultation. Instead, it simply ignores the occultation such that we do not take care of its effects on the target video object. A large area of the target object being covered will of course generate errors, especially when the object is covered so much that even humans can’t tell the object’s original characteristics.

Assume we take a video without noticing that there is a little drop of ink on the lens. This will cover part of the video up.

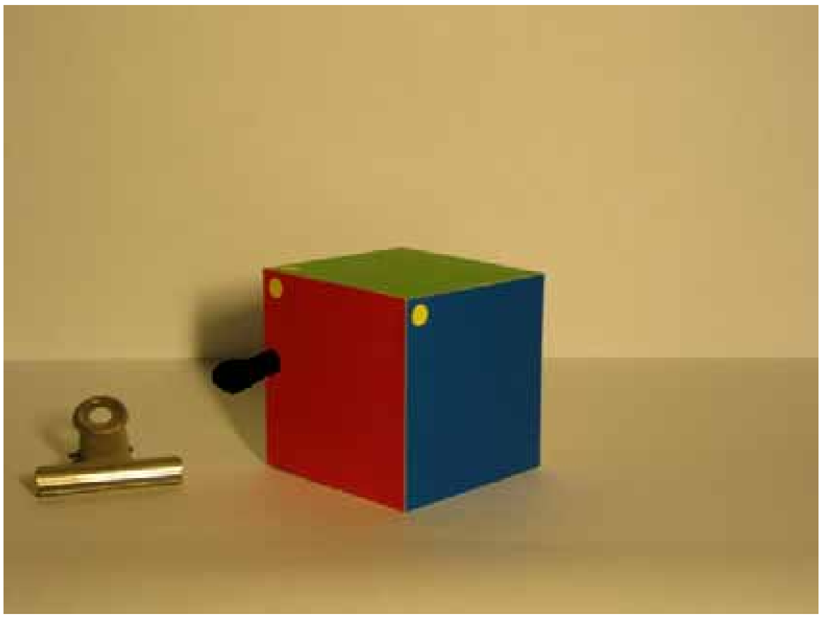


Figure 6.3.1 Careless photo taker…

Since the area of the red surface being covered is quite small, the occultation will almost have no effect at all.

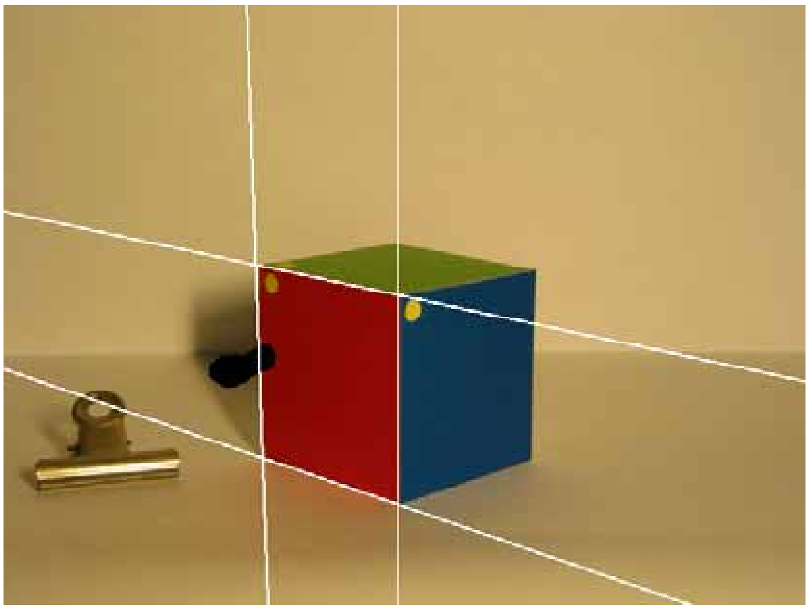


Figure 6.3.2 The square surface can more or less be figured out.

As mentioned before, large area of coverage will disrupt the information available on the surface for the computer to recognize. Now assume that the half of the lens cover remains at the front of the lens.

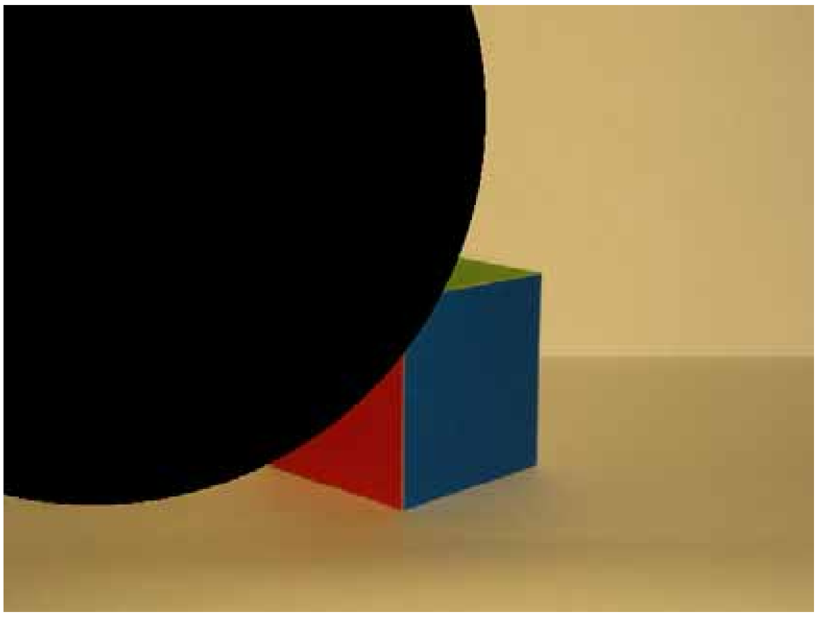


Figure 6.3.3 Really large obstacle…

Then, the system can do nothing to remove the occultation. Even if the computer knows it should be a square, it cannot estimate its size and how much the surface has been covered.

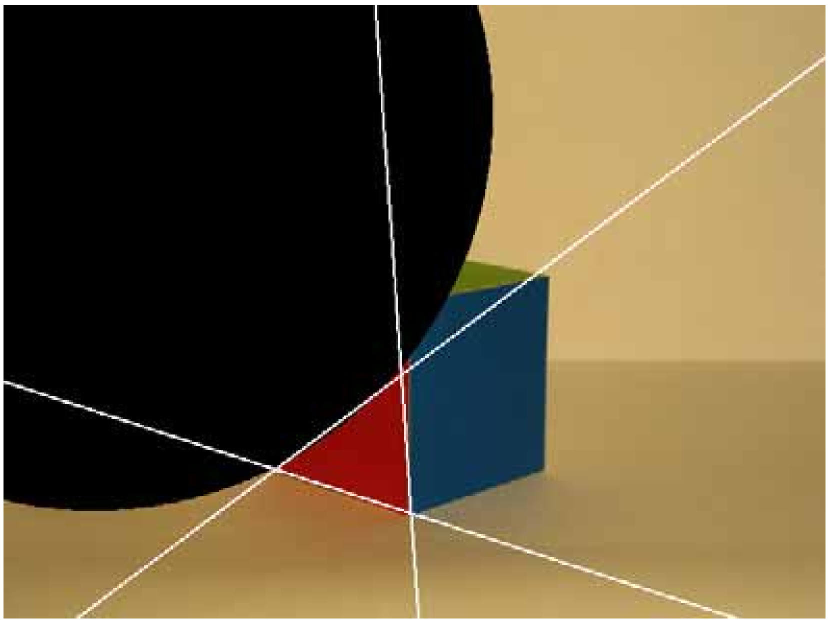


Figure 6.3.4 Half of a square equals a triangle!

We can see there are recognition errors due to the information loss of the surface.

Chapter 6

Simple Video Concept and Manipulation

6.1 Relationship between Consecutive Video Frames

It is well known that video files are large due to the number of video frames in them. A typical NTSC (National Television System Committee) standard will play a video at 24, 29.97 or 30 frames per second (fps), while a PAL (Phase Alternating Lines) standard will play at 25 frames per second. For these reason, even if we are dealing with a video clip of 10 seconds in Hong Kong (PAL standard), we must process 250 video frames. [14]

Now let's calculate how much data we need to process. Assume the video is in VCD format (352x288), amount of data to be processed is:

352 (width) x 288 (height) x 3 (RGB colors) x 25 (frames) x 10 (seconds)
= 76, 032, 000 bytes (or 72.5MB)

, which is a large amount of data. Bear in mind that the video is only 10 seconds long. The amount of data to be processed will even be greater when DVD standard is used (720x576). Intensive data manipulation is a heavy burden to a computer system, even with the processor speed doubling in less than two years.

Before being frightened by the large amount of data we are going to process, let's look at the following video frames first:

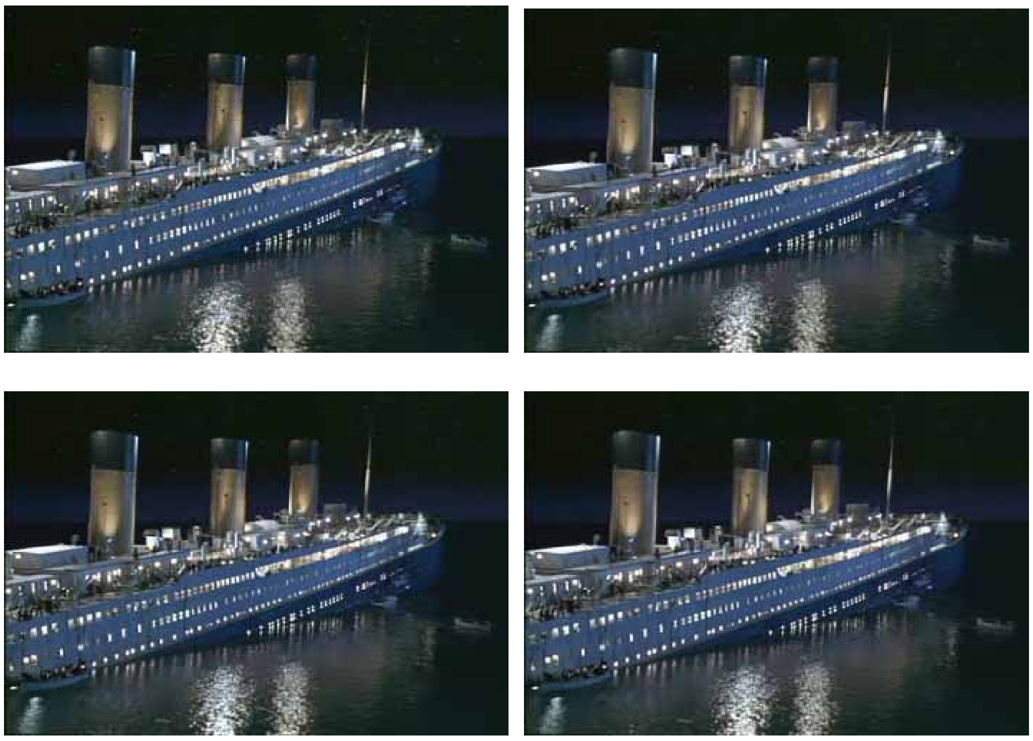


Figure 5.1.1 Four consecutive video frames from the film *Titanic*, note that the four images look almost the same.

The four video frames are almost identical to each other. In fact, the camera is moving from right to left for several frames. That means, by looking at the first two or three video frames, we can almost get the idea of the coming video frames until we meet the key frame (scene change). And, this is the main reason of introducing a simple translation detector to eliminate redundant calculations.

6.2 Translation Detector

The object can move in six directions relatively to the camera: To, from, up, down, right and left. For this reason, we do not need to scan the entire video frame in order to locate where the video object is.

It is inevitable to scan the entire image for the first frame since there is no information available for the computer at all. Once the first frame is being scanned, we can more or less get an estimated position of where to start the edge detection scanning, provided that the moving velocity of the video object in the video is not too high. This is the same as scanning a smaller image file.

Take a true example. If we have the following video frames:

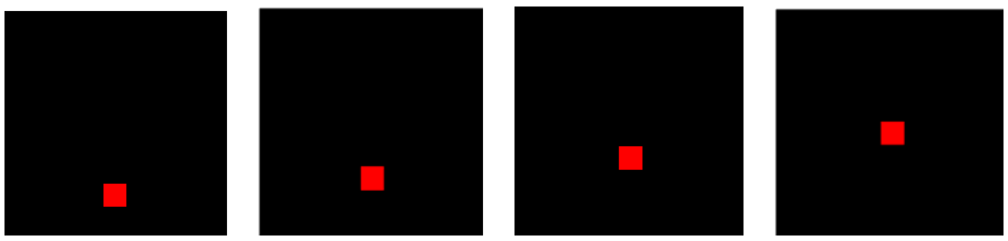


Figure 5.2.1 Bouncing red box

We can easily observe that there is a red square moving up the black background. Although the difference between the four frames is more obvious than the ones in the film, there are no other significant changes other than the position of the red square. So, we can mark down the uppermost and lowermost value of the coordinates where the video object can be found. For the latter video frames, scan only the part which is n pixels higher than the uppermost bound and n pixels below the lowermost bound, where n is a constant related to the velocity of the red square. Since the red square is very small in comparison to the whole picture, the effect of tracking the motion of the square is just like reducing the size of the whole picture by 1/10 or even more. Hence, the processing speed is greatly improved. This algorithm applies also when the object is moving from left to right or even moving towards the camera.

However, if the object move faster than expected, then the preset "bound" may not cover the object completely. Errors may occur if this is the case.

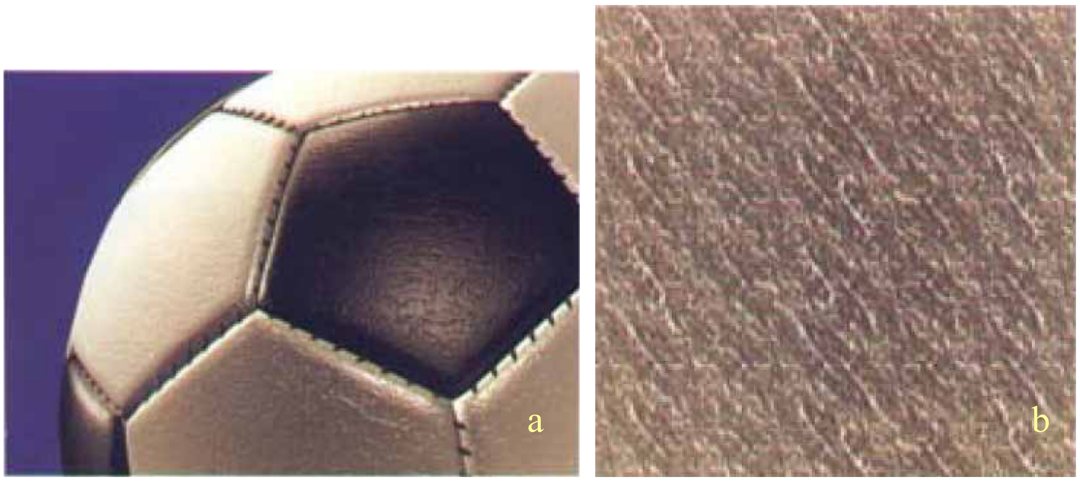
Chapter 7

Texture mapping

7.1 Background information

What is texture mapping?

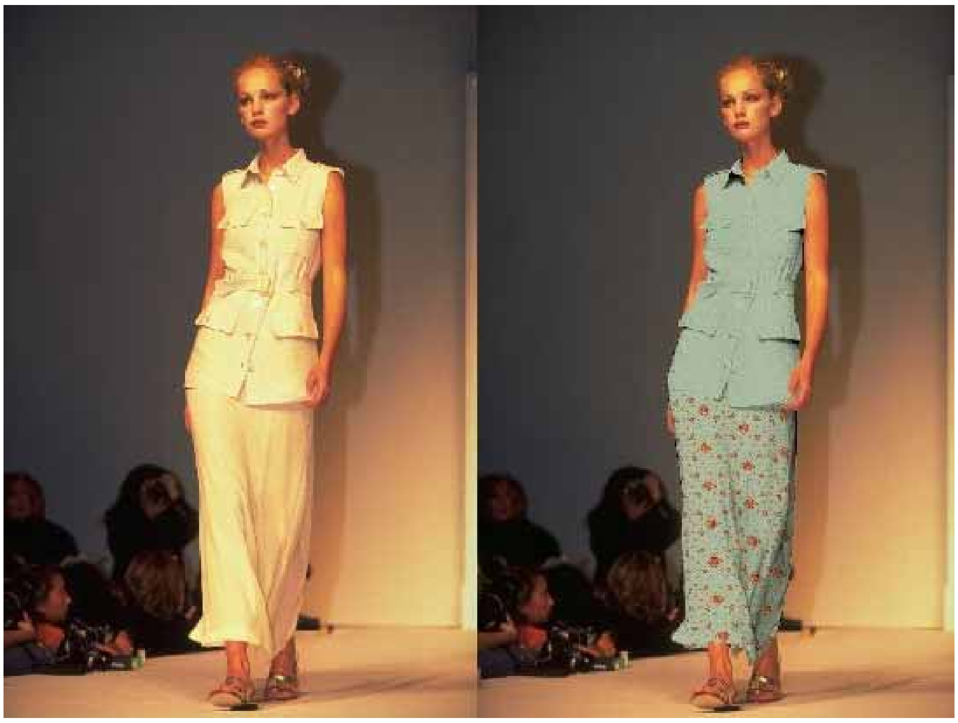
Texture mapping is a graphics design technique used to wrap a surface of a 3-D object with a *texture map*. After mapping a texture onto an object, the color of the object at each pixel is modified by a corresponding color from the texture. Usually, the texture map is an image so that after wrapping, the object will have a surface which looks like the texture image. More generally, the texture map can also be a 2-D array of altitudes or brightness values to control the shading of a surface. A good example is bump-mapping which transforms a smooth surface to a rough one with a texture map of various altitudes.



Bump-mapping: the rough surface of the football in (a) is achieved by mapping the rough texture map in (b) onto the originally smooth surface of (a)

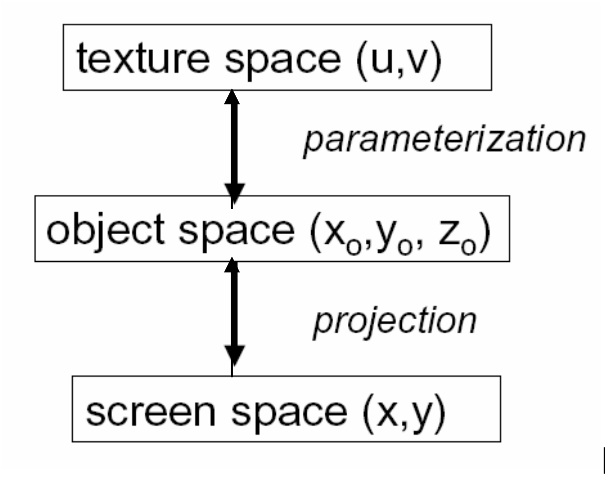
Modern applications of texture mapping

Texture mapping is a powerful technique for adding realism to a computer-generated scene. This can be applied to a wide variety of fields including computer games, medicine, molecular graphics, architecture, art and design, film production as well as fashion. In most of the cases, it saves time and money from the production of various objects which can actually be modeled by the computer. Its capability of creating realistic images and versatility in surface transformation makes it a widely used and developed technique.



Application in fashion: The same piece of cloth can be mapped with various patterns to determine the type achieving the best visual impact. [2]

Texture mapping model



[3]

In general, the process of texture mapping consists of two steps: mapping and projection. The two steps can be done interchangeably. Either we can map the texture pattern onto object surfaces, then to the screen space by projection; or we can do it the other way round i.e. mapping the pixel areas on to object surfaces, then to texture space. [4]

Our work

In this part of the project, we will focus on mapping a 2-D bitmap texture map onto a rectangular surface of a box, which can be in any orientation and shading. Note that we have skipped the middle part of object space mapping since 2-D plain surface mapping may not require 3-D consideration. This is a fundamental part of texture mapping which is essential to further techniques like sphere/cylinder mapping or 3-D texture mapping.

7.2 Definition of terms

Here are some of the terms related to texture mapping:

1. *Texture coordinates*
   These are usually represented by (*u, v*). They are the location in the texture map which contains color information for the image. [5]

2. *Image coordinates*
   These are usually represented by (*r, c*). They are the pixel location of the image, which is the rectangular surface of a box in our case. [5]

3. *Mapping function*
   It maps texture coordinates to image coordinates or vice versa. What it looks like depends on the shapes of the surfaces, as well as the actual coordinates of the texture map and the image. The mapping from texture to surface must be invertible, that is, every surface point gets only one color assigned, while it is good to have multiple surface points mapped to the same texture point. [6] Examples include linear mapping functions, linear scan-line interpolation and projective mapping.

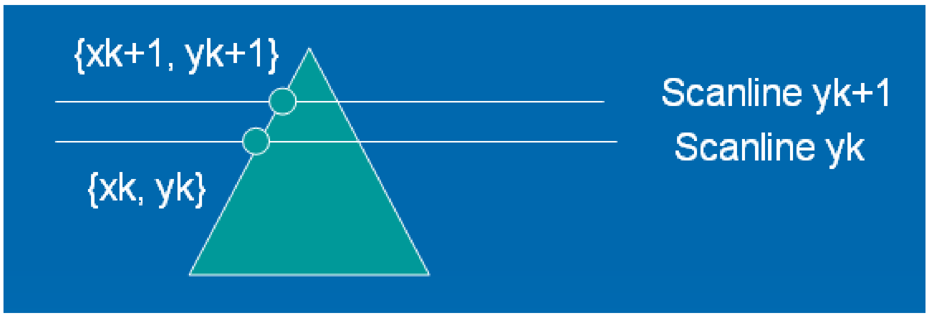
4. *Forward mapping*
   This describes the process of mapping the texture map to the image surface. A mapping function which maps texture coordinates to image coordinates is used. It is also called texture scanning.

5. *Inverse mapping*
   This describes the process of mapping the image surface to the texture map, which is the opposite of forward mapping. A mapping function mapping image coordinates to texture coordinates is used. It is also called pixel-order scanning.

6. *Scan-line conversion*
   This area-filling technique processes the image line by line and performs the necessary calculations on every pixel on the line. It can be applied to either forward mapping or inverse mapping with a particular mapping function. [5] This is an essential step since every pixel of the image surface needs to be processed to determine its corresponding texture coordinates.

7.3 Comparison of forward mapping and inverse mapping with scan-line conversion

Scan-line conversion

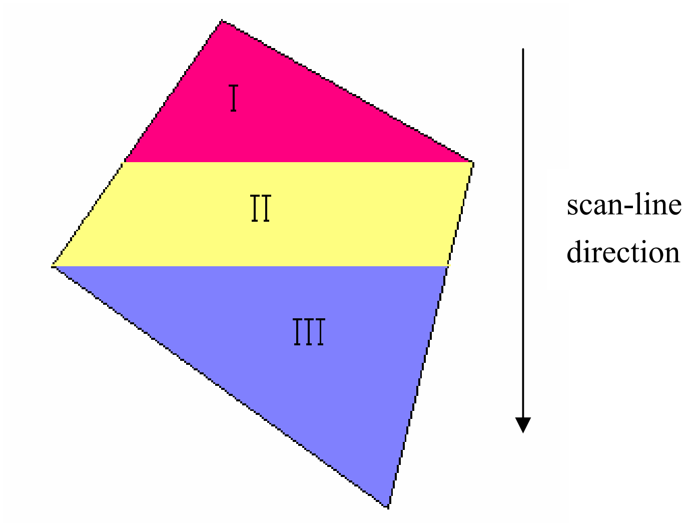


For line yk, all the pixels on the right of {xk,yk} inside the polygon will be scanned. After scanning the line yk, yk+1 which is one pixel above yk will be scanned.

This approach is typically used in general graphics packages to fill polygons, circles, ellipses, and other simple curves. Another approach is start from an interior point and paint outward till boundary conditions are met. This is useful for more complex shapes and hence we will use the scan-line approach. [4]

In the project, we need to scan a quadrilateral in any orientation and shape. The polygon needs to be partitioned into 2 or 3 sub-polygons for scanning due to a change in boundary conditions. For each sub-polygon, the boundary points are checked to see if they overlap with the vertices. If this condition is met, the corresponding polygon finishes scanning. To be more efficient, the boundary points of the line r+1 can be computed from line r by the addition of a small increment because the boundaries are straight. See the figure below.

Note that this scan-line algorithm is particularly useful for inverse mapping since the image surface to be scanned is likely a non-rectangular quadrilateral which is the case for the texture map. It also facilitates the use of interpolation technique as the mapping function to be described in the coming sections.



The quadrilateral is divided into 3 smaller polygons for scan-line filling
It is divided into 2 if the middle two vertices are on the same row

A look into forward and inverse mapping

Forward mapping and inverse mapping are two opposite concepts that one should choose to be used in texture mapping. A comparison of the two is as follows.

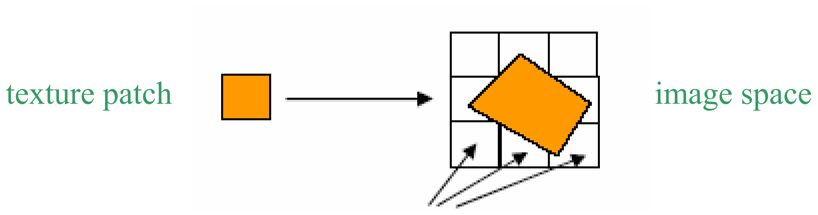
	Forward mapping	Inverse mapping
Principle	Map from texture to image	Map from image to texture

Algorithm	for u = umin to umax
	  for v = vmin to vmax
	    r = R(u,v) } Mapping
	    c = C(u,v) } function
	    copy pixel at source (u,v)
	      to destination (r,c)
		for (r,c) = polygon pixel
		  u = TEXR(r,c)
		  v = TEXC(r,c)
		  copy pixel at source (u,v)
		    to destination (r,c)

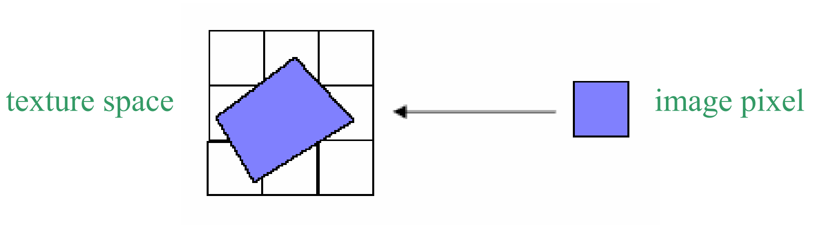
Ease of implementation	easy as long as the mapping function is known	more complicated as it involves scan-line conversion
Calculation of the fractional area of pixel coverage	Yes	No
Possibility of aliasing	Yes	Can be avoided by simple filtering or resampling

The mapping function of forward mapping can be obtained by projective mapping, which is a set of pre-computed solutions to the mapping function given the corner coordinates of the texture and the image (to be described). Though this involves overhead computation, the conversion can be quite effective after the coefficients of the mapping function are found. As inverse mapping involves scan-line conversion, scanning can be less efficient.

However, the major drawbacks of forward mapping are the required calculation of fractional area of pixel coverage and the possibility of aliasing. A selected texture patch usually does not match up with the pixel boundaries, requiring calculation of the fraction it contributes to a particular pixel. Thus, an entire image size accumulation buffer is needed to accumulate all the contributions to every target pixel. Inverse mapping, in contrast, avoids pixel subdivision calculations, and allowing anti-aliasing techniques like filtering to be easily applied. [4] [7]



The texture patch has fractional contributions to the color of the pixels pointed to by the arrows

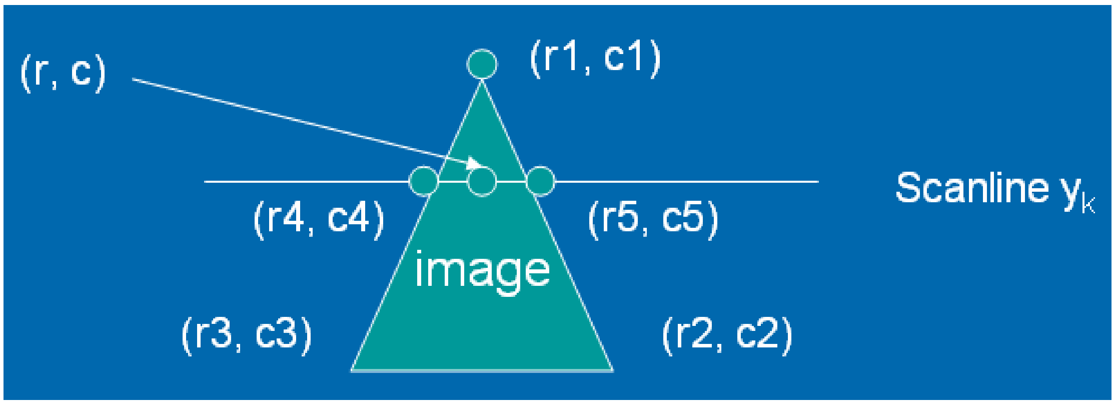


The color of the image pixel is computed as the weighted average of source pixels

In view of the above advantages of inverse mapping algorithm, we have chosen this as our mapping direction.

7.4 Types of mapping functions

Linear scan-line interpolation



This method uses inverse mapping and the concept of interpolation to determine the mapping function. To illustrate the algorithm, we assume that the texture coordinates (*u, v*) of pixel (*r, c*) are to be found, and that (*r1, c1*), (*r2, c2*) … (*r5, c5*) have the corresponding texture coordinates (*u1, v1*), (*u2, v2*) … (*u5, v5*). The image corner coordinates as well as the corresponding texture coordinates are given.

The algorithm consists of several steps [5]:

1. Determine s, the cutting proportion of (r4, c4) on the left boundary by

   $$(r4, c4) = s*(r1, c1) + (1-s)*(r3, c3)$$

2. Determine the texture coordinates (u4, v4) by

   $$(u4, v4) = s*(u1, v1) + (1-s)*(u3, v3)$$

3. Find (u5, v5) in a similar manner.
4. Determine t, the cutting proportion of (r, c) on the scan-line by

   $$(r, c) = t*(r4, c4) + (1-t)*(r5, c5)$$

5. Find (u, v) by

   $$(u, v) = t*(u4, v4) + (1-t)*(u5, v5)$$

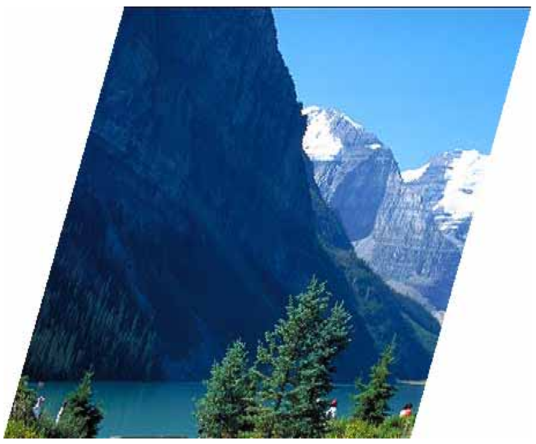
It would be too inefficient to apply the 5 steps consecutively for every image pixel. To raise efficiency, we compute (Δu, Δv) which is the difference of texture coordinates of neighboring image pixels on the same scan-line, say (r, c) and (r, c+1). When the next pixel (r, c+2) is computed for the texture coordinates, we just add (Δu, Δv) to the texture coordinates of (r, c+1). This is so since the mapping is linear.

Linear 2-D mapping with simple linear transformations

This is to apply a series of basic linear transformations to the texture map or the image map for texture mapping. Simple linear transformations include translation, scaling, shearing and rotation. Matrix mathematics show that each of them can be computed by pre-multiplying the source coordinates with a 2x2 matrix. Hence, the mapping function is made up of the multiplication of a series of 2x2 matrices, which is also a 2x2 matrix.

Note that this kind of mapping is capable of transforming a rectangular texture map into parallelograms in any orientation. It can also be very fast once the mapping function is found. The big disadvantage is that it cannot map to surfaces like trapeziums which are not parallelograms, resulting in flaws. This type of situation is very common for camera-taken image surfaces because of perspective considerations and camera distortions (to be described).

original texture map


after shearing and scaling

Projective mapping

This is the most general 2-D linear mapping that can map any quadrilaterals to any quadrilaterals [5]. It makes use of the general solution of a system of equations as the mapping function. Its principle is illustrated below.

$$u = (a_{11}r + a_{12}c + a_{13})/(a_{31}r + a_{32}c + 1)$$
$$v = (a_{21}r + a_{22}c + a_{23})/(a_{31}r + a_{32}c + 1)$$

where the 8 coefficients a_{ij} are to be determined.

Now suppose the texture map is a square with corner coordinates (0,0), (0,M), (M,N) and (N,0), while those for the image surface consists of (x1,y1) … (x4,y4). Then

Eqs. for corner #0	Eqs. for corner #1	Eqs. for corner #2	Eqs. for corner #3
$0 = \frac{a_{11}x_0 + a_{12}y_0 + a_{13}}{a_{31}x_0 + a_{32}y_0 + 1}$	$N = \frac{a_{11}x_1 + a_{12}y_1 + a_{13}}{a_{31}x_1 + a_{32}y_1 + 1}$	$N = \frac{a_{11}x_2 + a_{12}y_2 + a_{13}}{a_{31}x_2 + a_{32}y_2 + 1}$	$0 = \frac{a_{11}x_3 + a_{12}y_3 + a_{13}}{a_{31}x_3 + a_{32}y_3 + 1}$
$0 = \frac{a_{21}x_0 + a_{22}y_0 + a_{23}}{a_{31}x_0 + a_{32}y_0 + 1}$	$0 = \frac{a_{21}x_1 + a_{22}y_1 + a_{23}}{a_{31}x_1 + a_{32}y_1 + 1}$	$M = \frac{a_{21}x_2 + a_{22}y_2 + a_{23}}{a_{31}x_2 + a_{32}y_2 + 1}$	$M = \frac{a_{21}x_3 + a_{22}y_3 + a_{23}}{a_{31}x_3 + a_{32}y_3 + 1}$

With 8 equations and 8 unknowns, the coefficients can be determined as follows, which is the general solution to the system.

Let

$$\begin{aligned} \Delta x_1 &= x_1 - x_2 & \Delta y_1 &= y_1 - y_2 \\ \Delta x_2 &= x_3 - x_2 & \Delta y_2 &= y_3 - y_2 \\ \Delta x_3 &= x_0 - x_1 + x_2 - x_3 & \Delta y_3 &= y_0 - y_1 + y_2 - y_3 \end{aligned}$$

then

$$a_{13} = \frac{\begin{vmatrix} \Delta x_3 & \Delta x_2 \\ \Delta y_3 & \Delta y_2 \end{vmatrix}}{\begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}} \qquad a_{23} = \frac{\begin{vmatrix} \Delta x_1 & \Delta x_3 \\ \Delta y_1 & \Delta y_3 \end{vmatrix}}{\begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}}$$

$$\begin{aligned} a_{11} &= x_1 - x_0 + a_{13}x_1 & a_{21} &= x_3 - x_0 + a_{23}x_3 \\ a_{12} &= y_1 - y_0 + a_{13}y_1 & a_{22} &= y_3 - y_0 + a_{23}y_3 \\ a_{31} &= x_0 & a_{32} &= y_0 \end{aligned}$$

For the case of mapping a surface of a random quadrilateral to a square, some more steps are done after finding the coefficients a_{ij} as above. They are represented as follows:

Compute A'_{ij}

$$\begin{array}{lll} A'_{11} = a_{22} - a_{23}a_{32} & A'_{12} = a_{13}a_{32} - a_{12} & A'_{13} = a_{12}a_{23} - a_{13}a_{22} \\ A'_{21} = a_{23}a_{31} - a_{21} & A'_{22} = a_{11} - a_{13}a_{31} & A'_{23} = a_{13}a_{21} - a_{11}a_{23} \\ A'_{31} = a_{21}a_{32} - a_{22}a_{31} & A'_{32} = a_{12}a_{31} - a_{11}a_{32} & A'_{33} = a_{11}a_{22} - a_{12}a_{21} \end{array}$$

Finally compute A_{ij} by

$$A_{ij} = \frac{A'_{ij}}{A'_{33}}$$

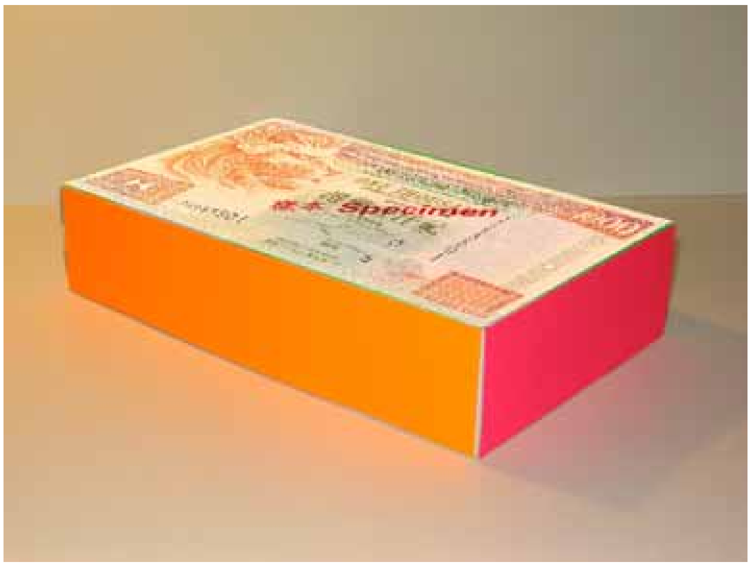
and replace a_{ij} with A_{ij}.

Combination of the above two cases results in a mapping from a arbitrary quadrilateral to another arbitrary quadrilateral. However, despite its versatility, the major disadvantage of the algorithm is it involves expensive computation [8]. When there are a large number of surfaces to be mapped with textures in this way, it can be very costly.

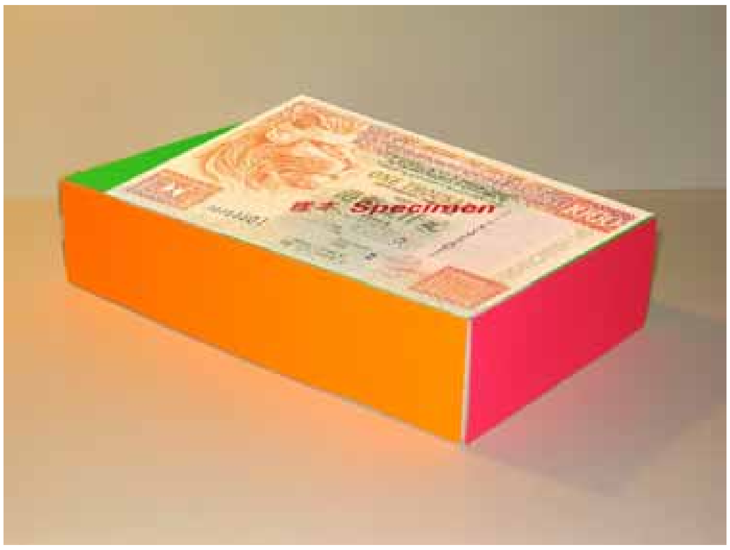
7.5 Comparison of linear mapping functions

	Linear scan-line interpolation	Linear mapping with simple linear transformations	Projective mapping
Speed	slow	fastest	slowest
Accuracy	high	poor	high
Ease of implementation	quite complicated	easy	acceptable

Although linear mapping with simple linear transformations is the fastest one, it has low accuracy and limitations like the possibility of aliasing. When the surface is regularly shaped such as a parallelogram or very small, it can be very useful. Linear scan-line interpolation and projective mapping have similar performance.



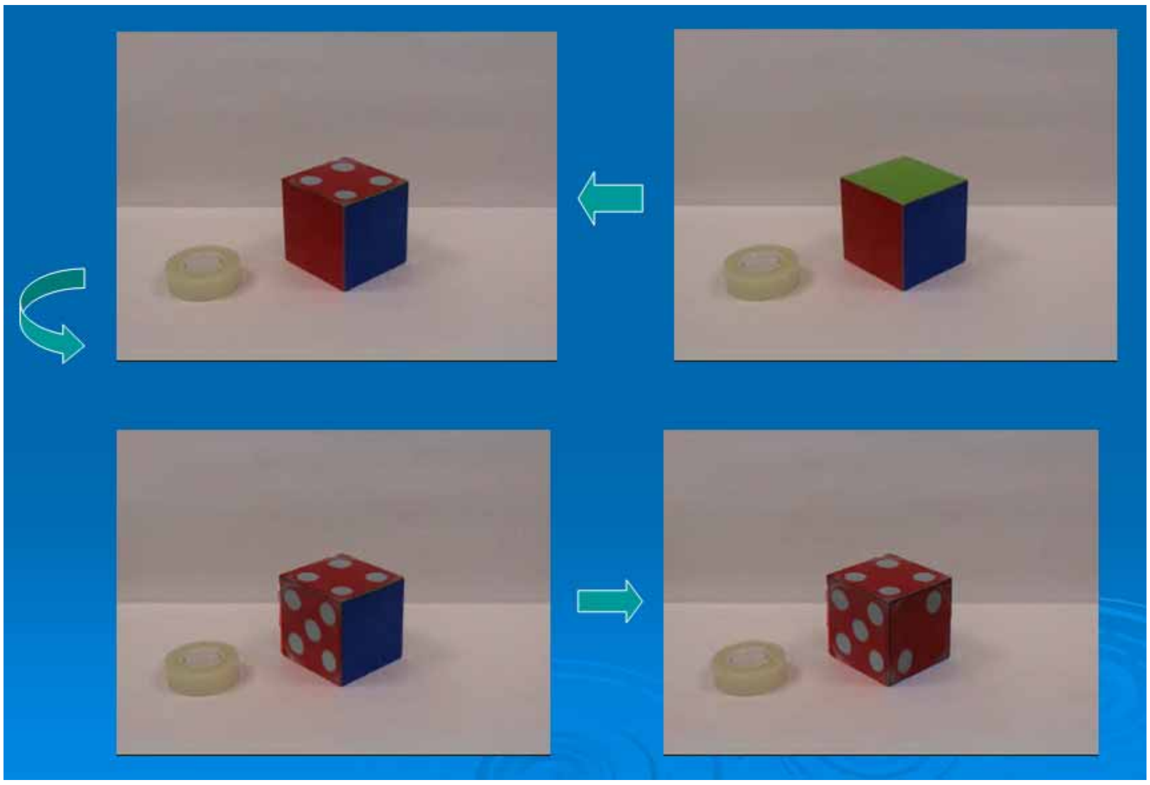
A box mapped with a thousand HK dollar note by **linear scan-line interpolation**



Usually, the image surface is not a parallelogram. Using **simple linear transformations** has low accuracy.

7.6 Multiple face mapping

This is to apply the texture mapping function consecutively on different faces so that the whole object will be covered with different textures to further enhance reality. The box can be modeled into a piece of brick, a pile of notes, etc which has all the faces mapped with textures.



The cube can be turned into a dice after multiple face mapping.

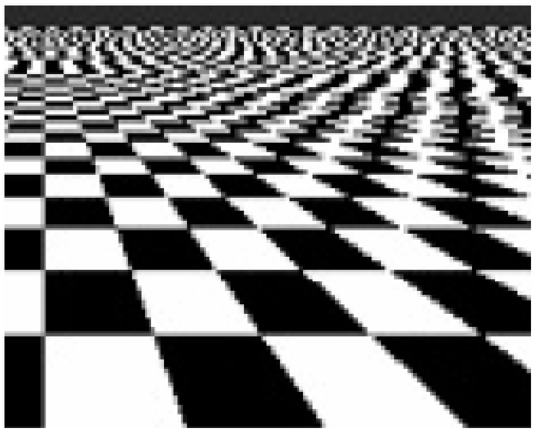
Chapter 8

Limitations of texture mapping

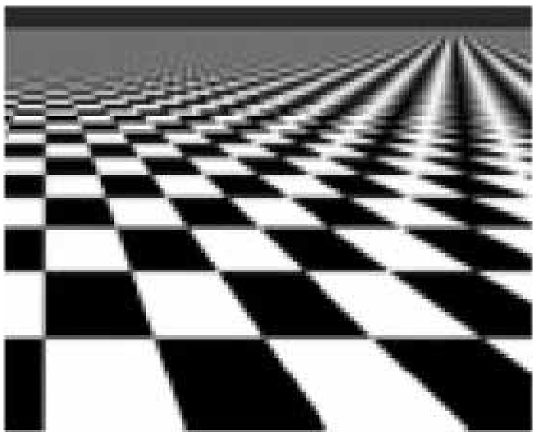
8.1 Possibility of aliasing

Illustration

This is the phenomenon that some displayed primitives have a jagged, or stairstep appearance [4]. Sometimes, even when the texture map does not have aliasing, the image surface after mapping can have. This unwanted artifact can happen during the stage of image acquisition, or during image mapping.



The upper part of the floor contains distorted shapes – aliasing



the floor pattern is so smooth after anti-aliasing operations!

Causes

Aliasing is mainly due to low-frequency sampling (undersampling). When the frequency of the texture map gets very high, we may not be able to sample it at sufficient points, resulting in unwanted discontinuities. Even drawing a line can have aliasing since continuous images always contain very high frequencies [5].

Anti-aliasing techniques

Theoretically, to avoid aliasing, the sampling frequency should be at least twice the highest frequency occurring in the object, referred to as the Nyquist sampling frequency.

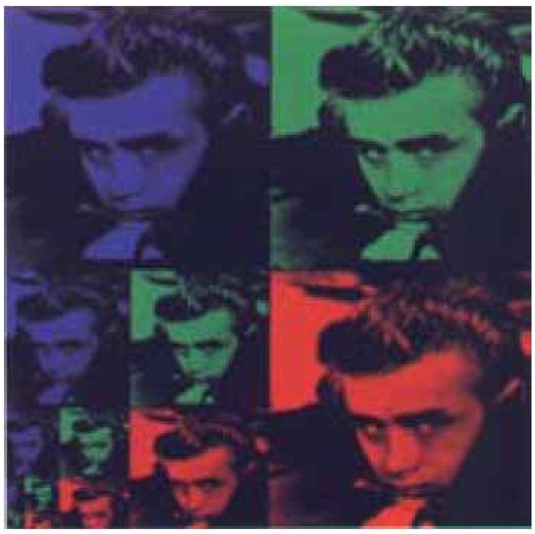
$$F_s = 2f_{max}$$

Some common methods of reducing aliasing include:

(i) Supersampling
This simply increases the sampling rate by treating the screen as if it were covered with a finer grid than is actually available. It is also called post-filtering

(ii) Filtering
Examples include Gaussian filter and a “box” filter. The high-frequency components are filtered and the highest frequency occurring in the image is lowered.

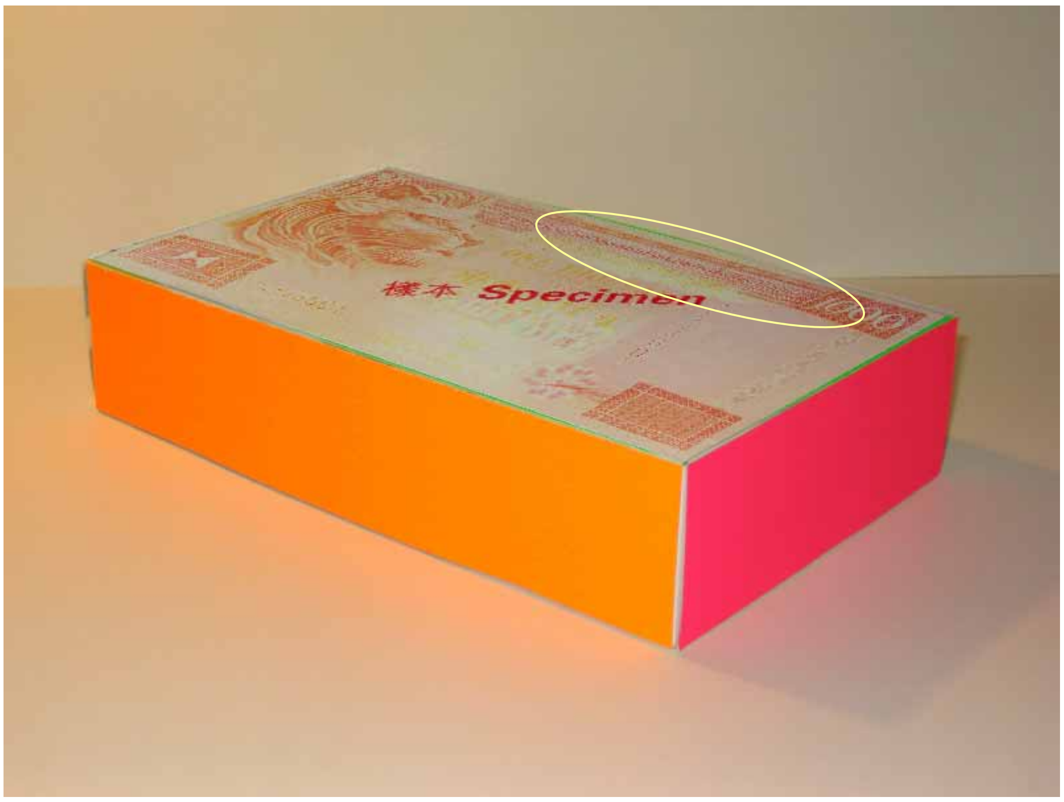
(iii) Mip-mapping
The texture map is pre-computed at different resolutions using filters. The pyramid of images generated can be efficiently stored. During texture mapping, the texture of the most similar size with the image surface is selected and mapped.


A pyramid of pre-computed texture maps in different resolutions

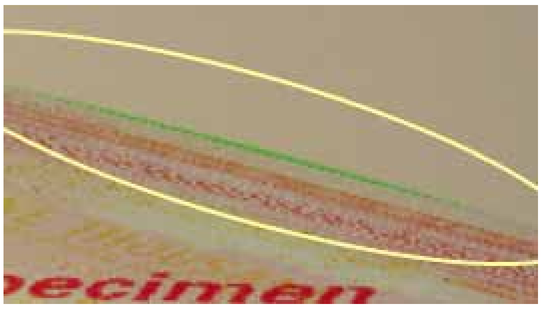
Note that aliasing is not a minor problem. When the texture map consists of repetitive patterns, aliasing can lead to a distortion of information to such an extent that the output image can probably be far from satisfactory.

8.2 Camera distortion

Not only is this a problem for edge detection but it also brings troubles to texture mapping. Our algorithm maps rectangular texture maps to arbitrary quadrilaterals with straight edges. But unfortunately, since the problem of camera distortion is virtually impossible to get rid of, the slightly bended edges result in mapping flaws.



Camera distortion results in flaws



Chapter 9

Improvement and Future Work

9.1 Better User Interface

Up to now, the system only works under a command line system. That means, users have to type in the following information by keyboard in order to make the system work:

- Name of image file to be processed.
- Number of times for the system to perform smoothing.
- Surface hue value of the target object.
- Surface saturation value of the target object.

We are proposing a graphical user interface for this system, so it will have the following advantages:

- Typo-proof. There is no problem for a user to mistype a value. The value can be corrected at any time before the execution. For command line system, a user has to start over when there is a typing mistake.
- Continuous work. The graphical user interface allows a user to process multiple videos without restarting the application. On a text-based system, the application exits once the first task is done.
- What-You-See-Is-What-You-Get (WYSIWYG). The graphical user interface provides an input window and an output window. A user can try and adjust different values of target hue/saturation and smoothing levels before he/she actually write the result to the disk.
- Increased performance. Usually a 32-bit application will have a better performance than a 16-bit text-based application.

We will make use of Microsoft Visual C++ MFC to develop the graphical user interface.

9.2 Faster Processing Time

Since the parts with the project are finished individually, they do not work at their optimal when they are combined together. Currently, processing a standard VCD frame still needs more than 2 seconds in average.

In order to achieve real time augmented reality (AR), we will try to improve the performance of edge detection and translation detection. We hope that by the end of the second term, the project can achieve at least 20 frames per second at standard VCD quality.

Possible areas of time saving:

- Using better algorithm of edge detection (we are adopting a simple algorithm with no optimization).
- Merge repeated works in different components into one.
- Try to remove the redundant equation processor part by improving the remaining part (the most challenging part). In fact, if the remaining part has a high accuracy in processing, there is no need for us to use an equation processor.
- Try to define a new bitmap format that stores the odd rows at the beginning, follow by the even rows just like the interlaced GIFs. This allows the system to get a quick picture before it finishes reading the entire image file.

9.3 Improve the Accuracy and Error Tolerance

Up to now, the system can only work well when the input image is sharp and the occultation is small. So, we planned to improve this by doing the following things:

- Design some special patterns on the pure color surface such that the computer knows how much or even which part of the object is being covered. In fact, the yellow spots on the cubes in section 6.2 and 6.3 are used to detect the orientation of the box. It is not complex enough to estimate the information about the whole surface.
- Improves the edge detection algorithms to further minimize the effect of occultation.
- Include constraints in equation detectors such that unreasonable equations cannot be derived (currently, only the number of sides is given as a constraint).

9.4 Integrated MPEG Decoder and Encoder

Our project can only process video frame sequences instead of a real video stream. That means, we must use a third-party application (such as VirtualDub) to extract all the video frames for us. After the process, use another application to combine the video frames back to a video stream. Since the project is still in the engineering stage, we will use this method to test our program until it is bug free. For future convenience, an MPEG codec will be included in the project such that the job can be done with only one executable.

9.5 Work on Universal Objects

Currently the system will only work on cubes or cuboids. Actually other objects can be tackled if conic equation can be derived. In the second term, we will add support of cylinders, pyramids and cones.

9.6 Automatic calibration

At present stage, the user needs to specify which face of the cube to be texture-mapped by clicking that particular surface as input. The system then analyses the RGB value of the pixel selected and detects similar pixels to determine the location of the image surface to be mapped.

To enhance convenience, calibration can be done beforehand so that the user doesn't need to click for surface selection. The idea is to take a photo of a color palette under the same lighting condition as the object of the image is in. The palette photo is then input to the system which then knows how different colors look like under that particular lighting condition. Hence, if a red surface is going to be mapped, the user does not need to select the surface since the system already knows the RGB values of red.

9.7 Generalization of texture mapping

Meanwhile, we are focusing on mapping the color and shading to an image surface. To make texture mapping more realistic, factors like shininess and transparency need to be taken into account. For example, a texture map of wood and one of glass certainly have different degrees of shininess and transparency. Pure consideration of color and shading is certainly not enough.

Chapter 10

Conclusion for the First Semester

In this project, a system for surface detection and texture mapping is proposed. Although it is still at a preliminary stage, we have already grasped the fundamentals of digital image processing and had a glimpse at what post-TV production is about. More specifically, we have learnt the following techniques for digital image processing:

1. Get more knowledge about bitmap file structure. The project requires a deep understanding about how bitmap files store the 2D pixels. Also, the cons and pros of the bitmap file structure are learnt.
2. The HSV color scheme. We have learnt that the HSV color scheme is more natural and easy to be applied on human-friendly algorithms.
3. Geometrical method of describing the relation between a set of points. In this project, we have learnt how to use Hough Transform Voting algorithm to estimate a linear equation that passes through a large number of points.
4. The method of Median smoothing. This provides an accurate smoothing of images.
5. Different types of mapping functions and algorithms, like linear line scan conversion and projective mapping.
6. Texture mapping involving patterns, brightness and related problems like aliasing and camera distortion.

This is only the first step to developing a more comprehensive and effective post-TV production system. In the future, definitely more will be done to strive for further development and enhancement of the system.

Chapter 11

A Brief Project Revisiting and Revision

11.1 Mission Accomplished in the Last Semester

In this project we are implementing simple software which can perform simple video-processing tasks like object detection and replacement.

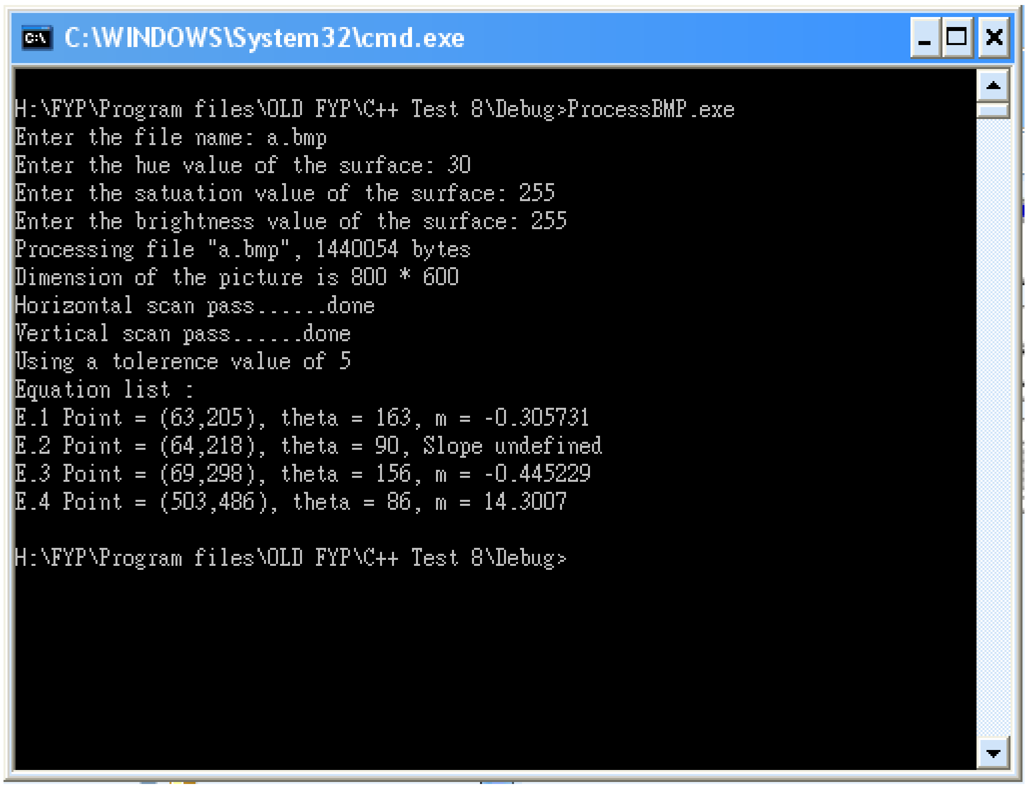
In the last semester we have implemented a version of software which can do the following things:

- Read in an image of pixel format
- Convert the color modal to HSV
- Detect and select 4 corner points from a given bitmap, which has a rectangular object in it
- Map a specific rectangular surface given 4 individual points
-

The main parts of the project are the followings:

- Bitmap reader / writer
- RGB<->HSV converter
- Edge detector
- Equation finder
- Equation processor
- Smoother
- Texture mapper

The project is a console program which accepts the input from the user through keyboard.



```
H:\FYP\Program files\OLD FYP\C++ Test 8\Debug>ProcessBMP.exe
Enter the file name: a.bmp
Enter the hue value of the surface: 30
Enter the satuation value of the surface: 255
Enter the brightness value of the surface: 255
Processing file "a.bmp", 1440054 bytes
Dimension of the picture is 800 * 600
Horizontal scan pass......done
Vertical scan pass......done
Using a tolerence value of 5
Equation list :
E.1 Point = (63,205), theta = 163, m = -0.305731
E.2 Point = (64,218), theta = 90, Slope undefined
E.3 Point = (69,298), theta = 156, m = -0.445229
E.4 Point = (503,486), theta = 86, m = 14.3007

H:\FYP\Program files\OLD FYP\C++ Test 8\Debug>
```

Figure 11.1.1 The inputs of the program is being typed in manually.

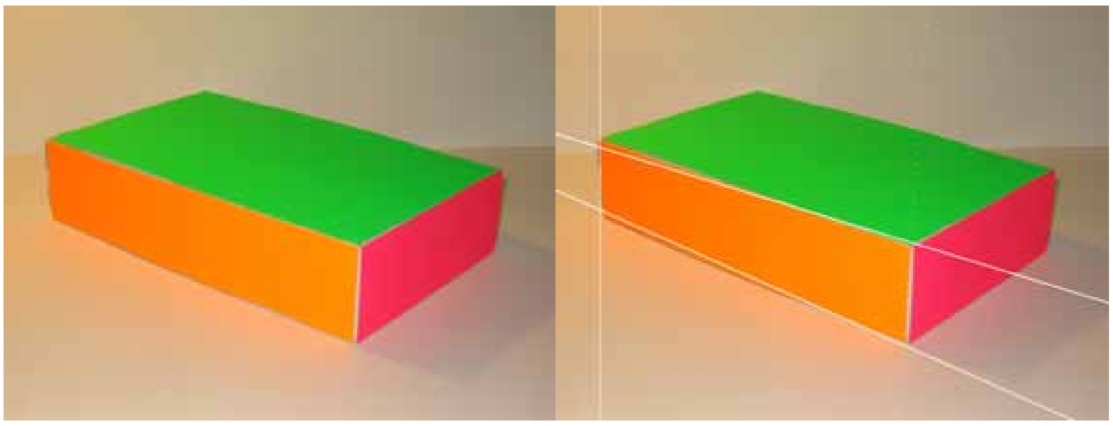


Figure 11.1.2 The target surface is being located through the use of four straight lines.

11.2 New Features Added

In order to get the goal nearer to the title of the project, a few new features have been added to the project.

The improvements which has been mentioned in the last semester:

- A graphical interface for the application. Users can map a texture to a given bitmap file or video clip file with only mouse clicks.
- Video processing support for both standard video playback and processing.
- Processed video can be saved to the disk drive for later use.
- Cylindrical surface mapping support means more objects can be processed.

The improvements which are added without mentioning:

- Improved detection speed by using additional markers which can be found directly.
- Any changes made to the bitmap files can be undone. Users do not have to use a new file and start all the things over.
- Support of multiple formats of video files, including MPEG and AVI.

11.3 The Main Components Added to the Project

In order to implement the newly added features, we have included the following parts in our projects:

- An MFC®-based [15] graphical user interface (GUI) core.
- Video clip file reader and writer
- Video frame processor (modified frame-based bitmap processor)
- Direct corner detector
- Enhanced texture mapper which supports more functions
- Equation analyzer

11.4 The Part Which Has Been Modified

In addition to the newly added parts, we have also modified some parts of the project such that the whole project works more efficiently.

- The bitmap I/O interface. It has been splitted into two parts – file reader and file writer – in order to work with MFC. Also, the performance of this part has been improved.
- The edge finder now processes the pixel map as a long array of pixels instead of a 3D array (Width x Height x HSV). This can increased the performance and decrease the memory usage.

- The original texture mapper has been modified such that it can now truly represent the light reflection of the original surface. This is called shaded mapping.
- Removing redundant equations has always been a work of high cost. For the reason of additional performance increase, we have removed the equation processor from the project. The original function of equation is integrated into other parts like bitmap I/O and edge detector. Also, the accuracy can be improved by the use of corner finder, so it is not a need to keep the equation processor part. The calculation of intersection points of the edge lines (I.e. the corner points) is left for a new class to work with. This is the equation analyzer. Instead of trying to remove extra equations from the equation list, the equation analyzer just draw out facts about the equations and returns them. The corner point is one of the examples.

We will look at the parts in detail in the following chapters.

Chapter 12

The Graphical User Interface

12.1 Why Need a Graphical User Interface?

We all know that a graphical user interface (GUI) is not the theoretical part of the project. That means the project can go without it. So, why we need a month to develop the GUI? The reason is:

- It is more user-friendly (at least to the developers, i.e. we). The process results can be gathered by clicking the mouse button for just a few times.
- What-You-See-Is-What-You-Get (WYSIWYG). For processing images and video clips, it is important for the users to see what the outcome is in a short time. For ordinary users they can try out the preferred result more quickly. For developers like us, it will be useful in debugging.
- Users can handle multiple input files using GUI. Thanks to the multitasking capability of modern GUI operating systems like Microsoft Windows, we can open several programs in the main window in order to process them simultaneously or just compare the output.
- The previous input of the user is kept in the dialog boxes, such that user can use the previous input to perform the process on another bitmap or video clip file in the minimal amount of time.

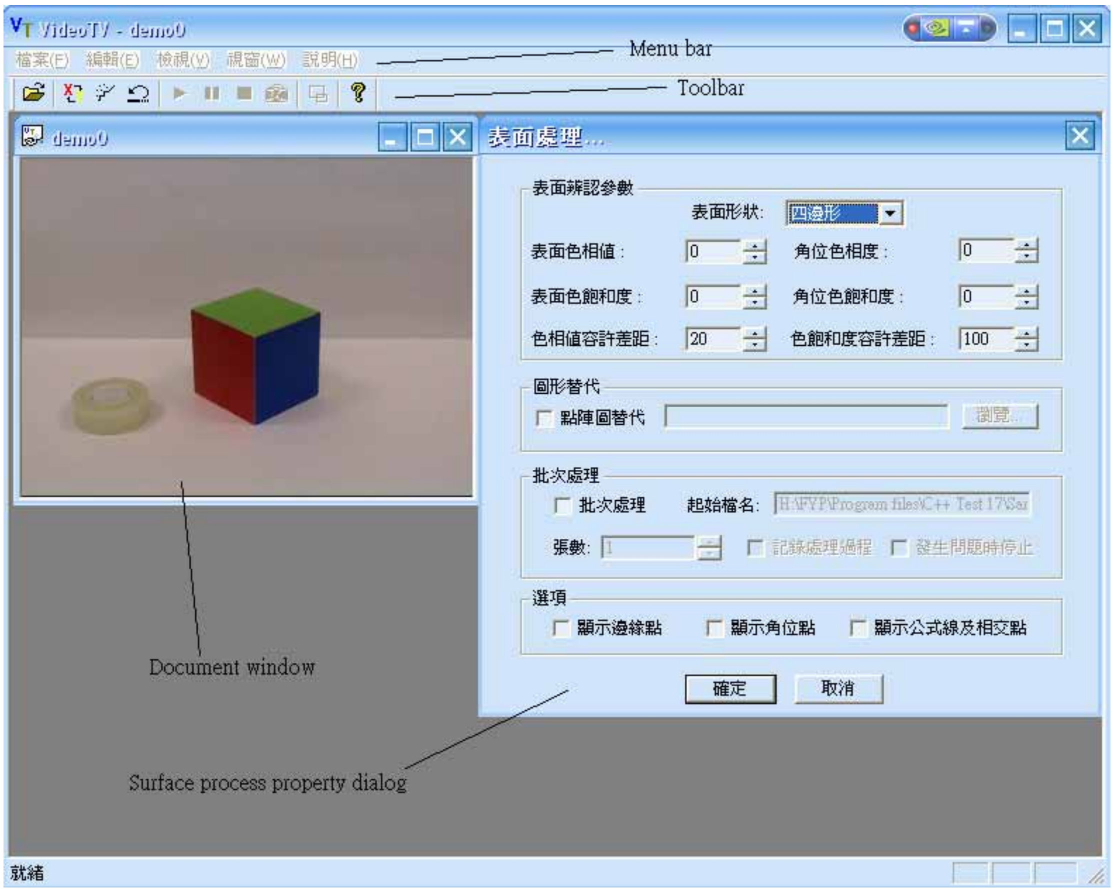
12.2 The Appearance of the Proposed User Interface

The graphical user interface has the following parts:

- Main window. This is the basic workspace.
- Document window. One of such window will be opened for each bitmap or video clip.
- Bitmap processing property page.
- Video processing property page.

- Bitmap smoothing page.
- Video smoothing page.

Below is an overview of the graphical user interface:


VideoTV - demo0
檔案(F) 編輯(E) 檢視(V) 視窗(W) 說明(H)
Menu bar
Toolbar
demo0
Document window
Surface process property dialog
就緒

表面處理...

表面辨認參數
表面形狀: 四邊形
表面色相值: 0
角位色相度: 0
表面色飽和度: 0
角位色飽和度: 0
色相值容許差距: 20
色飽和度容許差距: 100

圖形替代
☐ 點陣圖替代 瀏覽...

批次處理
☐ 批次處理 起始檔名: H:\FYP\Program files\C++ Test 17\Sar
張數: 1 ☐ 記錄處理過程 ☐ 發生問題時停止

選項
☐ 顯示邊緣點 ☐ 顯示角位點 ☐ 顯示公式線及相交點

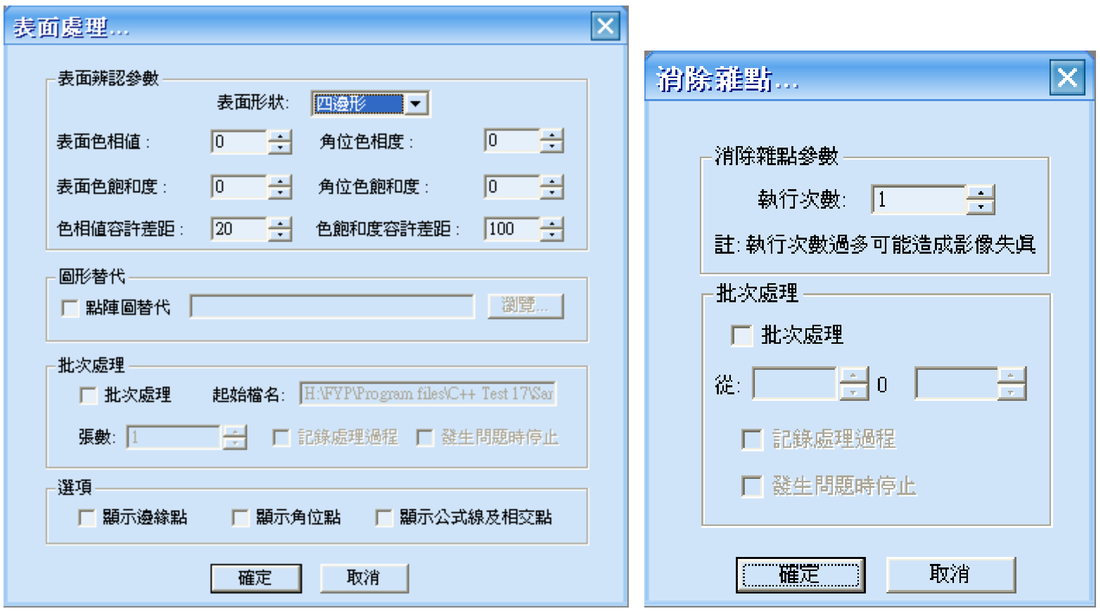
確定 取消

Figure 12.2.1 The graphical user interface

As mentioned in the last chapter, the graphical user interface is being developed with the Microsoft Foundation Classes (MFC), as this is the easiest way to migrate a text-based C++ program to a GUI one.

Also, the functions which appeared in the text user interface program can be packed together and link to a single dialog class, making the source files much easier to be managed.

Since the video processing will have more options for the users to input, the relative input parameter dialog boxes will be slightly different from the ones with bitmap processing.


表面處理...

表面辨認參數
表面形狀: 四邊形
表面色相值: 0 角位色相度: 0
表面色飽和度: 0 角位色飽和度: 0
色相值容許差距: 20 色飽和度容許差距: 100

圖形替代
☐ 點陣圖替代 瀏覽...

批次處理
☐ 批次處理 起始檔名: H:\FYP\Program files\C++ Test 17\Sar
張數: 1 ☐ 記錄處理過程 ☐ 發生問題時停止

選項
☐ 顯示邊緣點 ☐ 顯示角位點 ☐ 顯示公式線及相交點

確定 取消

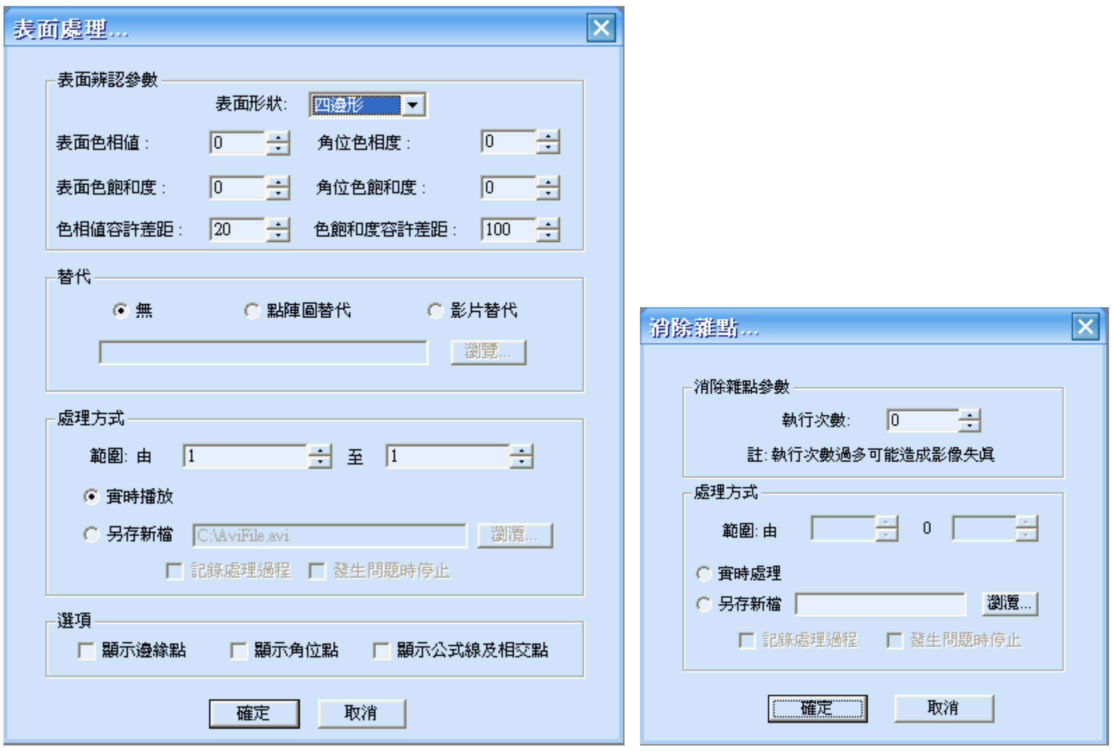
消除雜點...

消除雜點參數
執行次數: 1
註: 執行次數過多可能造成影像失真

批次處理
☐ 批次處理
從: 0
☐ 記錄處理過程
☐ 發生問題時停止

確定 取消

Figure 12.2.2 Process property page for bitmap processing


表面處理...

表面辨認參數
表面形狀: 四邊形
表面色相值: 0 角位色相度: 0
表面色飽和度: 0 角位色飽和度: 0
色相值容許差距: 20 色飽和度容許差距: 100

替代
[✓ 無] 點陣圖替代 影片替代
瀏覽...

處理方式
範圍: 由 1 至 1
[✓ 實時播放]
另存新檔 C:\AviFile.avi 瀏覽...
☐ 記錄處理過程 ☐ 發生問題時停止

選項
☐ 顯示邊緣點 ☐ 顯示角位點 ☐ 顯示公式線及相交點

確定 取消

消除雜點...

消除雜點參數
執行次數: 0
註: 執行次數過多可能造成影像失真

處理方式
範圍: 由 0
實時處理
另存新檔 瀏覽...
☐ 記錄處理過程 ☐ 發生問題時停止

確定 取消

Figure 12.2.3 Process property page for video processing

The graphical user interface uses a document-and-view multiple document interface (MDI) [16] structure to handle the opened documents. That is, each document is opened in an individual window (view) that is independent of each others. Again, the document windows of bitmap processing and video clip processing have different designs. The child window for video files has a seeking bar at the bottom, which means the users can search to a particular position for playback and processing.

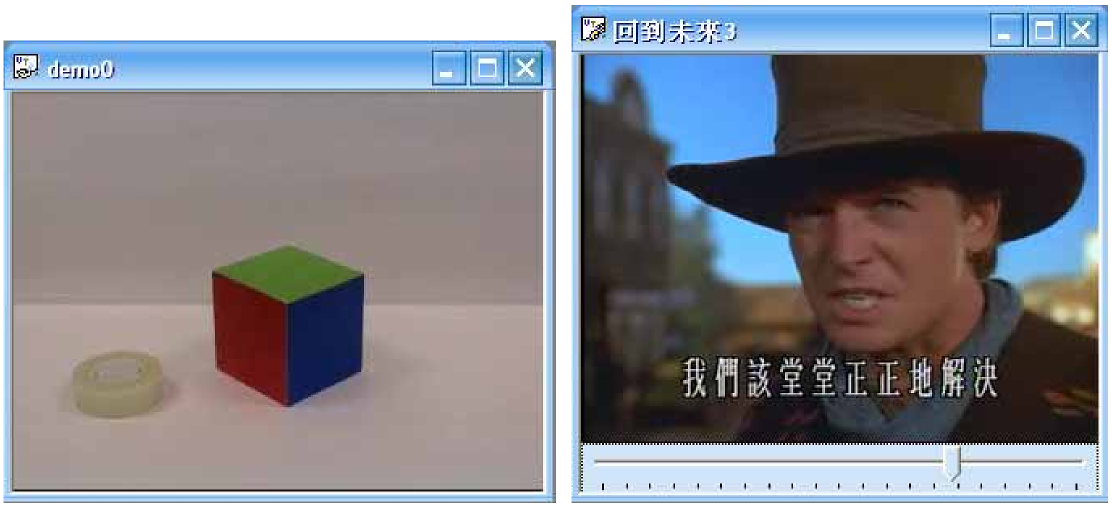


Figure 12.2.4 Document windows of a bitmap file and a video clip file

For user’s convenience, simple menu bar and a handy toolbar is also provided in order to accelerate works. Note that the menu bars for the two types of documents are also different.

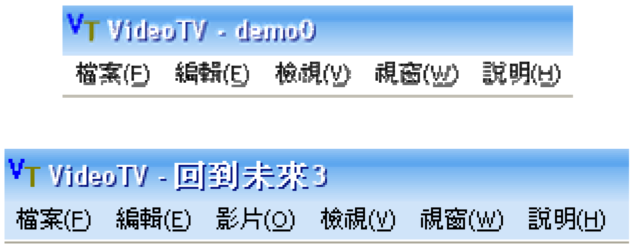

VideoTV - demo0
檔案(F) 編輯(E) 檢視(V) 視窗(W) 說明(H)
VideoTV - 回到未來3
檔案(F) 編輯(E) 影片(O) 檢視(V) 視窗(W) 說明(H)

Figure 12.2.5 Menu bars used for bitmap files and video clip files

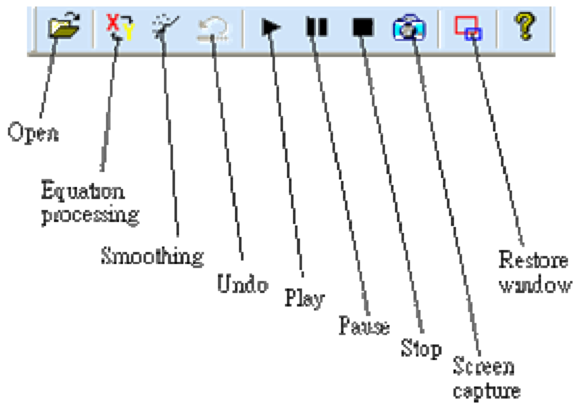
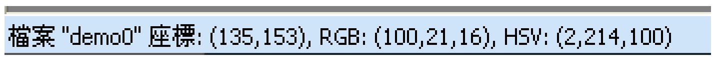


Figure 12.2.6 Toolbar used in the project

There is a status bar below the main window. It indicated which file user is now working on. Also, it provides color information of the current bitmap, which enables the user to get information of color for a specific pixel.



檔案 "demo0" 座標: (135,153), RGB: (100,21,16), HSV: (2,214,100)

Figure 12.2.7 Status bar at the bottom of the main window

The method of entering the colour information of the target surface is also easier. Users only have to left double-click on the target surface, and the HSV information will be passed to the surface processor automatically. When the users have right double-clicked the surface, it means that the corner labeling stickers' colour has been assigned. This feature is handy to process the image files, since the measurement of colour values by eyes or other software can be difficult and inefficient.

Since the video clip frames are being drawn onto the child window inside the graphics card, the colour of a video frame cannot be detected directly. So, users have to make a sample screenshot of the video frame first and then select the target colour value from the screenshot instead. Usually a sequence of video frames will contain images of similar colours, capture a screenshot once should be more than enough.

Chapter 13

Entering the World of Video Processing

13.1 Video Standards in the market

For television and other home video systems, the video standard can be divided into three categories, namely NTSC (National Television System Committee, used in North America and Taiwan), PAL (Phase Alternating Lines, used in West Europe and Hong Kong) and SECAM (Systeme Electronique Couleur Avec Memoire, used in France, East Europe and Russia). These standards differ mainly in the video resolution, refresh frequency and the way signals are transmitted in the atmosphere.

For computer video files, there are a lot more video standards than the television system. Besides home entertainment, digital videos have many other uses such as video conferencing. So, different standards are required to meet the individual needs of their applications.

The MPEG (Moving Pictures Experts Group) [17] standards are designed specifically for home video playback since they offer balanced compression ratio and quality at the cost of asymmetric encoding (the encoding process is much slower than the decoding process). Typical examples are MPEG1 (used to replace VHS tapes, resolution at 352x288 for PAL, data rate around 300kB/s), MPEG2 (high quality video playback on DVD discs, resolution at 720x576 for PAL, data rate around 1,000kB/s) and MPEG4 (high compression computer video with variable data rate, around 100kB/s for 640x480 VGA video).

On the other hand, the H.2xx standard is mainly designed for video conferencing purpose. The time for encoding is more or less the same as the time for decoding (i.e. symmetric encoding). But, the compression ratio and quality is usually lower than that of MPEG standard. A typical example is H.261, which is a video coding standard published by the ITU (International Telecom Union) [18] in 1990 for video conferencing over ISDN (Integrated Services Digital Network) 64kb/s lines.

From the above description we can see that the main difference between the various video file formats is the kind of compression. If a program can support these compressions than other problems should be easy to be solved.

13.2 Ways to Tackle with Different Video Standards

As we cannot predict which kind of video format the project is going to process, multiple format support must be added to the project. However, it is quite impossible to read through the documents about all the current video standards and put them into the project. Also, even if the system can be implemented in this way, it is certainly that there will be plenty of bugs and performance problem.

Fortunately, under Microsoft Windows we can use a set of predefined library and header files called DirectShow [19]. DirectShow is a member of Microsoft DirectX [20] starting from DirectX 8. It is for streaming media on the Microsoft Windows® platform. DirectShow provides for high-quality capture and playback of multimedia streams. It supports a wide variety of formats, including Advanced Systems Format (ASF), Motion Picture Experts Group (MPEG), Audio-Video Interleaved (AVI), MPEG Audio Layer-3 (MP3), and WAV sound files. In this project we will only make use of its video reading ability.

A DirectShow application will contain three major parts [21], namely:

- Input filter (Source filter), which prepares the data for the whole project.
- Processing filter (Transform filter), which changes the content of the input data.
- Output filter (Render filter), which writes the output to disk or display the result on the screen.

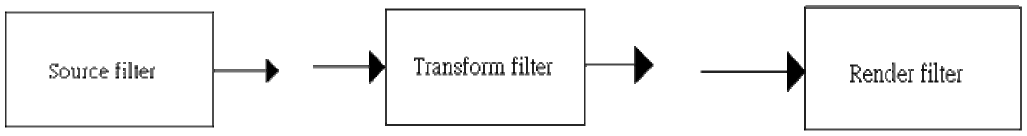


Figure 13.2.1 Three kinds of filters found in a DirectShow application

The use of DirectShow filters has the following advantages:

- The filters are part of COM models. That means they can be considered as individual parts in the process of development and debugging.
- As the filters are COM objects, they can be shared between different applications.
- The filters are usually registered with the operating system. Program which needs the filters can load them in runtime. That means the program itself can be smaller.

In this chapter we will talk about the source filter.

13.3 The File Input Source Filter

When designing a DirectShow application, a "block design diagram" should be prepared beforehand. This diagram arranges all the filters in an ordered way. In DirectShow, this diagram is called the "filter graph" [22].

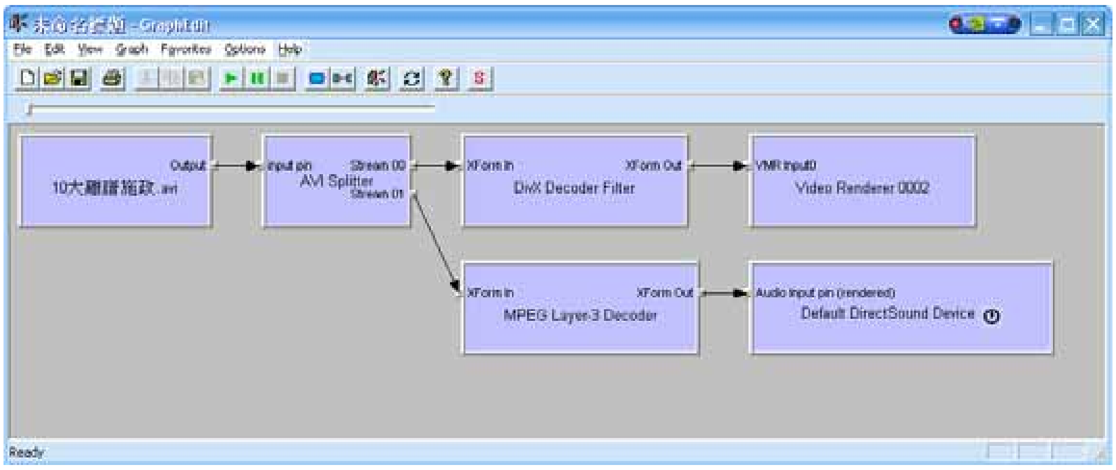


Figure 13.3.1 A typical filter graphic for rendering an AVI (Audio-Video Interleaved) file by Microsoft GraphEdit.

From the above filter graph, we can see that the source filter is an AVI video file. This is a file source filter which allows the project to read the data from a file. Besides file source filter, there are other kinds of source filters like URL source filter and Device source filter. The latter one will be useful and may be implemented in the near future since it allows the project to process real-time captured video from webcam, digital video camera and etc.

The file source filter requires only one input parameter – the file name. The file name is gathered from the user through the graphical user interface and is transformed into wide characters (WCHAR Unicode) beforehand. This procedure is required, since sometimes the file name may contain some non-English characters like Big-5 Chinese characters in the above filter graph example.

Besides AVI files, the acceptable file formats for the file source filter include MPG files and WMV (Windows Media Video) files.

Although the file source filter can accept any kind of video file format, we do not recommend users to feed in video files with very high compression ratio (especially those compressed with MPEG4 encoders like DivX [23] and XviD [24]). Although the video file can be opened, the decompression time may be too long, affecting overall performance of the project. The recommended file formats are RAW (no compression), MJPEG (motion jpeg), MPEG1 and MPEG2.

As the file source filter is a part of DirectShow, the program requires the computer to have DirectX9 or later to be installed.

Chapter 14

Further Improvement in Video Processing

14.1 Integration of Previous Work with Microsoft DirectShow

In the last chapter we have mentioned there are three kinds of DirectShow filters. Usually, the processing tasks should take part in the transform filters. However, we have changed the design and leave the part inside our program. This is because the purpose of the program is specific and it might not be used with other programs. Even if our program is built into a transform filter, the act may be meaningless since the resulting filter will be very big and cannot be loaded quickly as it is coded with the program itself.

The current way we make our program core to work with the input filter is through a method on the display render filter called GetCurrentImage(). This method will save the screenshot of the current position image in a byte array. To perform the video processing, we just get the pixels in the byte array, process it like what we did to the bitmap pixel bytes in the last semester and save the result to the disk, and advance the frame pointer by one frame. The way of saving frames to output files will be discussed in the later chapter.

After the pixels are being fed into the project, they are being processed in the same way as processing a single bitmap.

14.2 Improvements in Frame Reader and Writer

After implementing the graphical user interface, we know that bitmaps are interpreted as a single byte array instead of a 3D array. So, we have changed the original readers to suit the MFC structure.

Originally the text user interface uses the standard iostream library for data reading and writing. Now the CFile class is used instead to allow a closer system integration.

Since the reading and writing components are being reused by different parts of the project, they are separated from one another to reduce redundancy.

14.3 The Corner Point Finder

In the last semester we have proposed a method to locate the corners of a given surface. That is, to use a Hough-transform like statistic method to group the edge points and links them into a straight line. This method is useful if one of the corner or edge is being covered by some obstacles.

However, if the obstacle is very small or there is no obstacle, there is no need for us to perform such statistical grouping. This is because when the target object is getting bigger and bigger, the number of edge points will become very large such that computation for the edge line equations will take a very long time. Also, when the number of edge points increase, it is probable that a lot more noise points from the surroundings and the accuracy of the texture mapping will be seriously affected.

So, we should try to decrease the number of considerable points as few as possible. In the project we can make some assumptions, like we are going to map a rectangular surface. For rectangular surfaces, we can put some brightly coloured stickers at the corners such that they can be detected by the program. As the stickers are very small, a lot less points will be considered and this can significantly increase the surface searching speed.

The direct corner searching part uses more or less the same algorithm as that in the equation finding part, except it now uses a K-Means [25] like statistic instead of the Hough Transform statistic method. Firstly, the program will scan through and find out all the points which have the predefined colour of the corner stickers. Then, the program will perform the data clustering to group the selected points into as few as 4 groups (in the case of mapping rectangular surface).

Below shows the procedure of the data clustering:

- We first define the maximum distance that is allowed for the points to be grouped into one group. For example, we assume all the target objects are at least 20 pixels in its dimension so we set the maximum distance to be 20 (pixels).
- Randomly pick out 4 points from the selected ones. If any 2 of them are within

one another's maximum distance radius, group them into one group and pick out another point from the remaining ones till we have four distinct groups.

- For the remaining points, pick them out one by one and calculate the distance between the centroids of the individual groups (average of the coordinates of the points in that group). Put the point into the nearest group. Repeat the process until there is no more points left.
- If a point cannot be grouped (distance to any group is greater than the maximum distance), the point is put aside and we increment the void point counter by 1.
- Finally, we use the centroids of the four groups as the draft position of the corner points.
- If the number of groups is less than 4 or the value of void point counter is too large, that means the corner point detection is not successful and we will switch back to the original corner point locating algorithm.

We now illustrate the idea by several pictures.

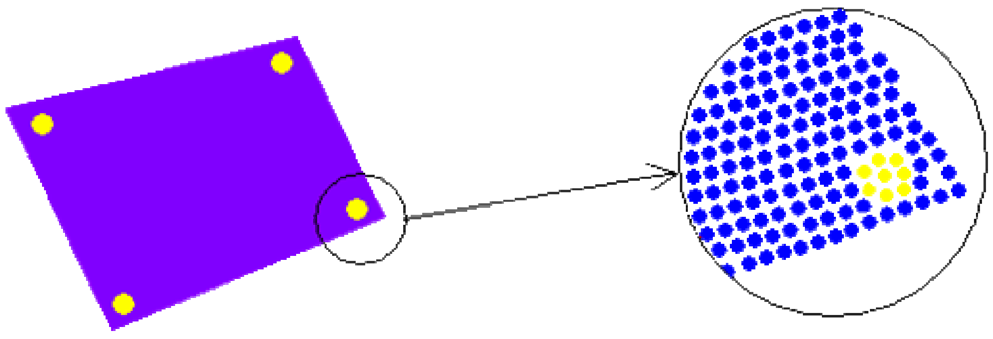


Figure 14.3.1 A simple illustration of a "corner point sticker" being fed into a computer system. Note that the size and positions of the points may not be exact in the above diagram.

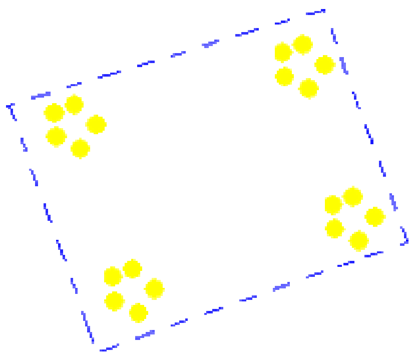


Figure 14.3.2 For simplicity we will assume each corner sticker will only have 5 points in it.

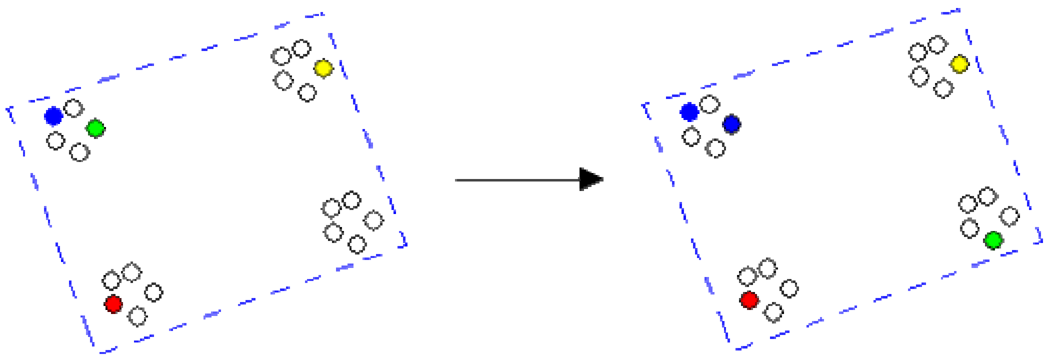


Figure 14.3.3 Randomly pick out four corner points. If two of the selected points are too close to each other, we group them together and choose a new point until four groups are formed.

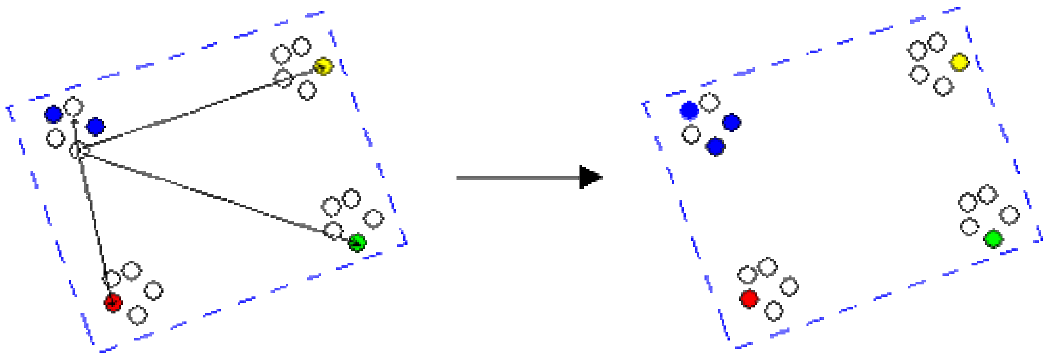


Figure 14.3.4 For the remaining points, calculate the distance between the point and the four groups. Group it to the nearest group.

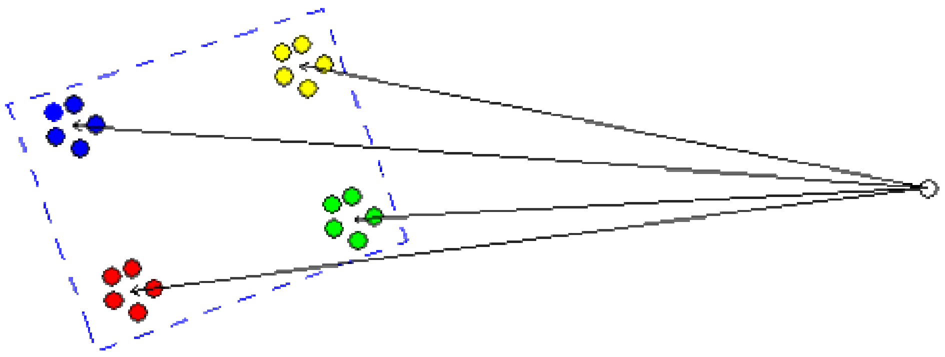


Figure 14.3.5 If a point is too far away from all of the four groups, it may be just a noise point and we should discard it.

By using this method, we can locate a drafted position of the corner points in a much shorter time. This is because there are far less corner marker points than the edge points. Also, only comparing the colour once is enough for the algorithm (once for comparing the color of the corner sticker), instead of twice for Hough Transform statistics (one for comparing the target colour and another for comparing the colour with its neighbour).

Readers may find out that the corner points calculated are slightly away from the true corners. This is because the centroids are being used instead of the point in the group which is the farthest away from the centre of the surface.

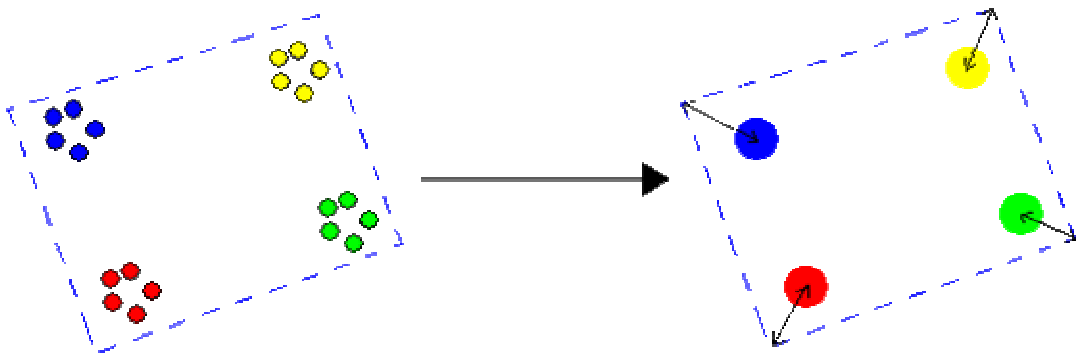


Figure 14.3.6 Use of centroids may introduce gaps between the computed corner positions and the real corner positions

Fortunately, most objects in video clips are regular in shape. I.e. they are either rectangular or square in shape. We can make use of this fact and "move" the computed points outwards a little bit. The direction of movement is known since we can calculate the "centroid of centroids", and just move the points away from this centroid should be okay.

However, it is difficult in this stage to calculate the real offset. That is, by how much should the computed corner points move outwards? In fact, this problem will happen if the object is moving from a very far position or vice versa. In daily life regular shaped objects seldom move in this way. We can just assign an arbitrary number to the offset, given that the number is more or less the same as the real distance between the corner stickers and the true corners. This method will, hopefully, give an acceptable set of results.

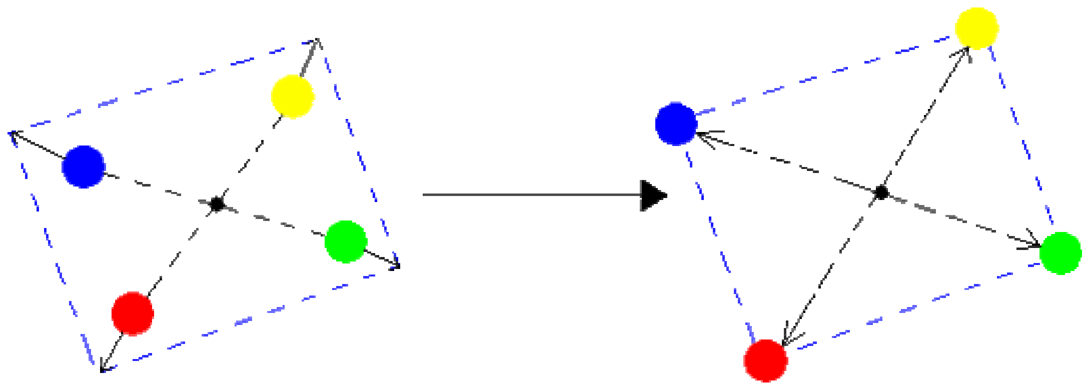


Figure 14.3.7 The computed corner points are moved out in order to give a more realistic result.

Now let us look at a real sample. Before the picture processing, we have the following picture:

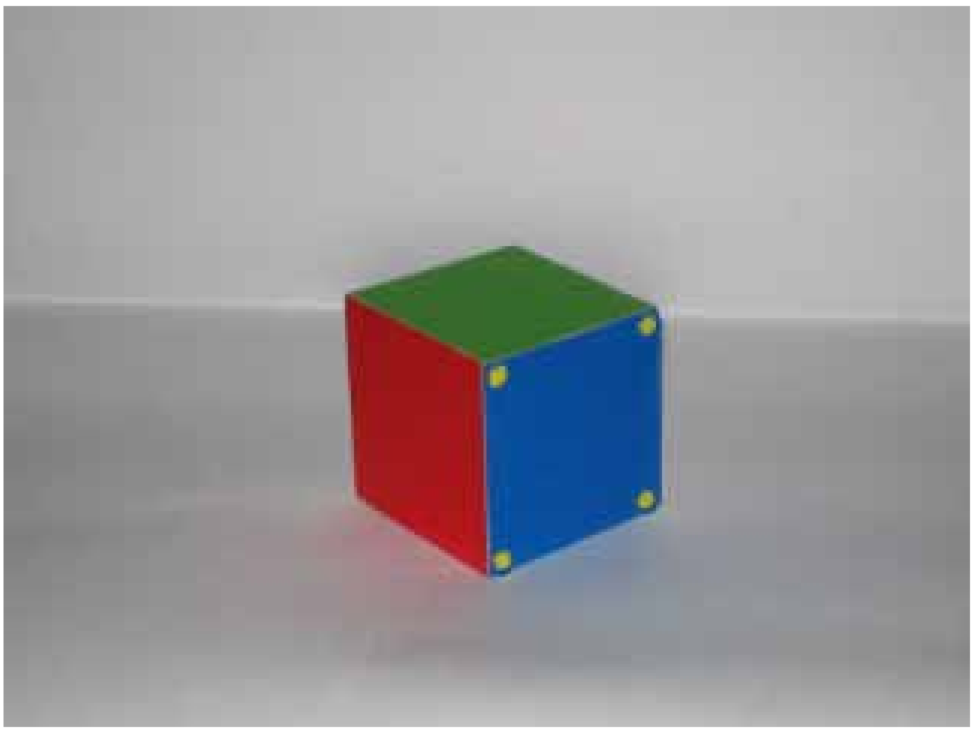


Figure 14.3.8 A box with its blue surface marked. I.e. the four corners are being labeled with yellow stickers.

Now we apply the new corner detection method on the picture. The result is show in the following figure:

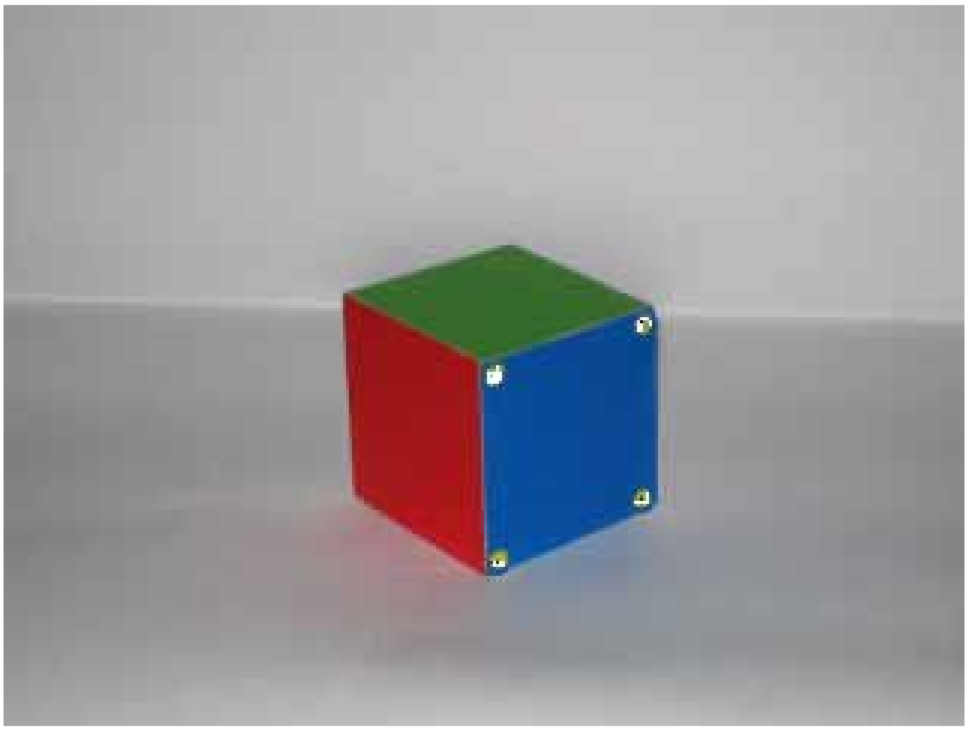


Figure 14.3.9 The box after process. We replace the detectable corner points with white colour (originally yellow) and draw a black dot at the group centroids.

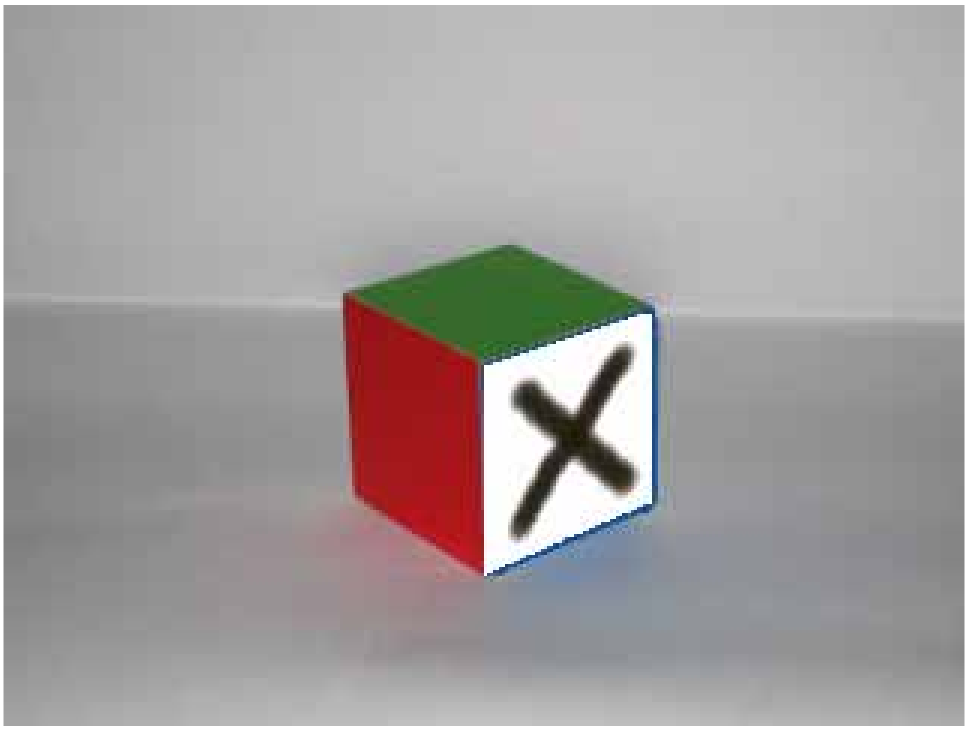


Figure 14.3.10 We try to map a texture bitmap file to the surface. Note that the mapping have some inaccuracy but it is acceptable.

14.4 Edge Detector

In the original design of edge detector, the pixel scanner just try to compare the colours and determine where sharp change of colours takes place (i.e. the edges). Now the edge detector will scan for the corner points at the same time. Since the scanning of corner points is not directional dependent (scanning vertically and horizontally will give the same result), we just implement it in the horizontal scanning pass.

14.5 Video Playback Control

If the project just has a video source filter, the video being read is not usable until there are some parts which can "play" (more precisely should be "run", since the playback rate may not be the same as the one specified in the video header) the video file to a position and get the screenshot at that moment. To do this we need to include two more parts in the project – the media control part and the media seeking part.

The media control part gives the project ability to run through a video file from the beginning to the end. It also provides important information about the video to the project core, such as the duration of the video, duration of each individual frame and etc.

The media control part only allows the user to play the file at constant rate, which is defined in the video itself. However, this is not suitable to our project program since processing video at the normal playback rate of 25-30 frames per second is far away from the processing power of the current project program. So, we need to use the media seeking part in the project. By using this part, the program can "jump" to any video frame specified by the seeker. This also means that the program is able to get the position of the next frame after it has processed the current one, without notifying the media control part to stop and wait for the process.

When the position of the next video frame is known, the project program will be able to invoke the GetCurrentImage() function on Windowless Control 9 in VMR9 rendering filter to return an array of pixel bytes.

Detail about the VMR9 rendering filtering filter will be discussed in the next chapter.

Chapter 15

Displaying the Result

15.1 Real-time Display

After processing the input video clips, the output must be shown to the user or otherwise the program is useless (at least practically useless). So, how we are going to show the result?

There are two kinds of methods. The first one is real-time display. That is, to display the output directly on the computer screen. This part hasn't been implemented completely yet since the performance of processing is not enough (10 frames per second for 320x240 video). Real-time processing will give a good visual result if and only if the processing speed can reach 15fps or more.

However, some parts have been prepared in the project in order to support the feature in the near future. In the project, we make use of the Video Mixing Renderer 9 (VMR9) in the DirectX 9 package to draw the video frames on the screen. The usage of Video Mixing Render 9 is:

- It makes use of the newer display card GPUs (graphical processing units). Since display hardware is involved in the process, the playback of video can be faster and clearer.
- It supports windowless mode, which means the result can be displayed onto any window surfaces provided that the dimension, position and parent window of the destination window rectangle are given. This can be useful in situations where we need to display the output in other areas of the program (e.g. open a new window to display the source and output video at the same time.

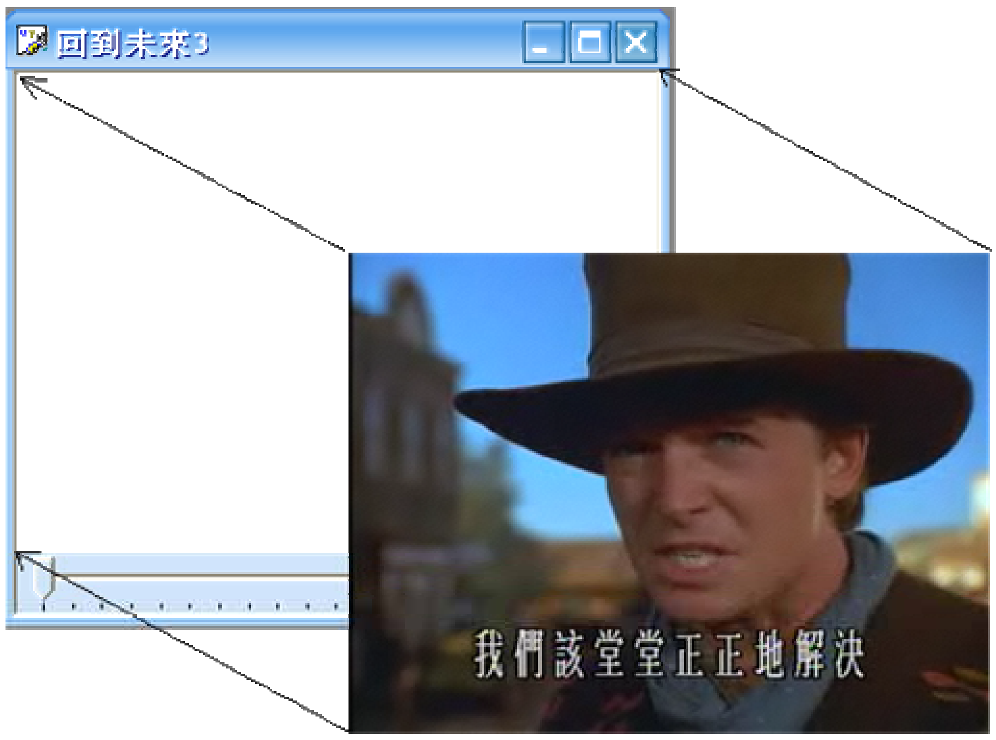


Figure 15.1.1 Video Mixing Renderer 9 can display the video source to anywhere with a given parent window frame and displaying window rectangle.

- It supports dynamic video frame capturing. If the video is being played or run, ordinary software video renderer cannot give a reliable result. This problem will, however, never happen on a VMR9 renderer.
- It supports the mixing of several video streams and displays the result on the screen (and hence it got its name). This is the most important feature of the VMR9, since it means that after the input video has been processed, we do not have to write the changes to the original files (this is important for real-time applications). Instead, we just display the processed part and display them altogether on the screen, and the users will think that the original video has been processed. In fact, the original video is totally unchanged.

Video Mixing Renderer 9 is a rendering filter which comes with Microsoft DirectX 9. So, any computer which has DirectX 9 installed should be able to run the project program.

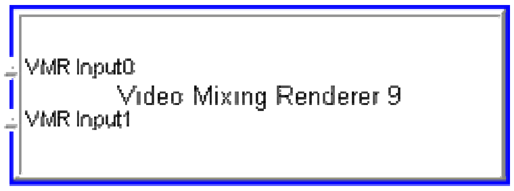


Figure 15.1.2 The representation of Video Mixing Render 9 filter in the Microsoft GraphEdit.

By using Video Mixing Renderer 9, it is also possible to develop a program which supports the development of video texture mapping. That is, to map a live video onto another video or bitmap just as if itself is a texture.

Video Mixing Renderer also works well with other DirectX components. For example, by working with Direct3D it can map a 3D mesh onto a specific surface.

15.2 Non Real-time File Saving

Currently, the way we use to preserve the output result is to save them to a file.

Video are a long sequence of images, and so are the output results (since the program will process the input video clip frame by frame). Hence, we can save the output image frames to a single file. After saving all the output image frames, the video file can be compressed with a specific encoder to decrease the video size. To achieve this goal, we have included a third-party module, *AviFile*, which can help us to implement this feature quickly.

Since the audio data can be separated from the video data, this part of the project does not preserve audio data in the output video file (we only offer this part with the output image frames).

The encoder for the output video can be arbitrary as long as it has been registered to the operating system. In our case, we will choose RAW (uncompressed) and DivX as the output format. Saving the file as RAW AVI files will be faster, while saving the file as MPEG4 compressed DivX AVI will give a much smaller output file.

Later on we will implement a version of AVI writer on ourselves which can let people to choose an encoder during runtime.

Up to now, we can draw a complete structural diagram for our project.

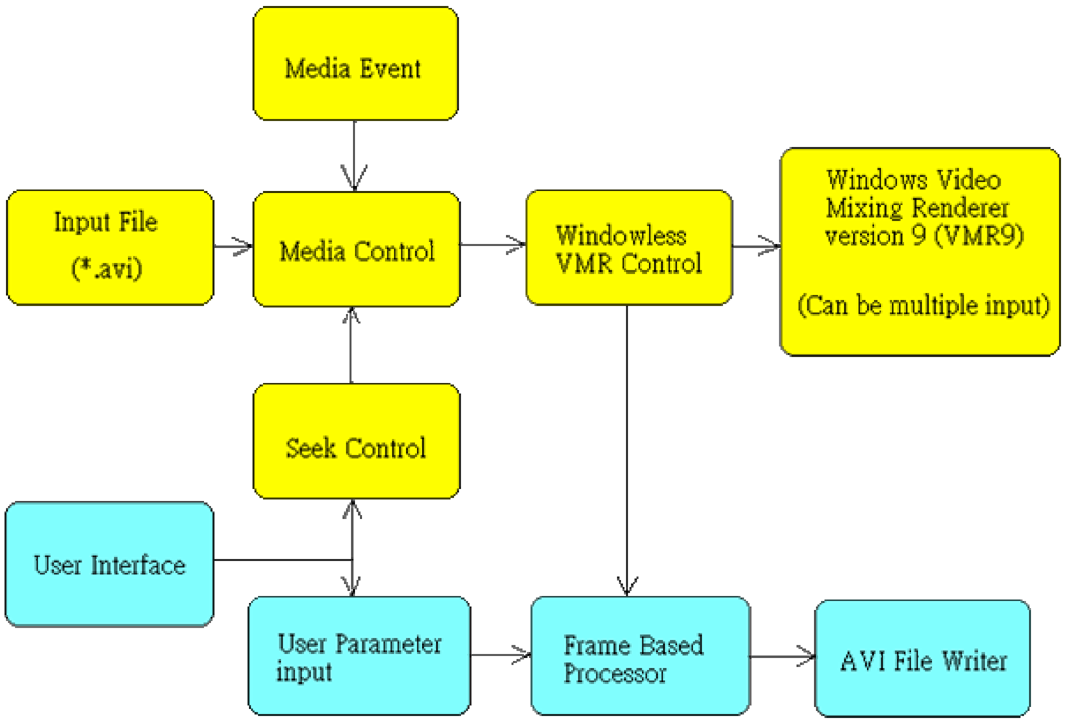


Figure 15.2.1 The overall system structural diagram. The yellow parts are implemented by DirectShow classes.

Chapter

16

Cylindrical Texture Mapping

16.1 Introduction

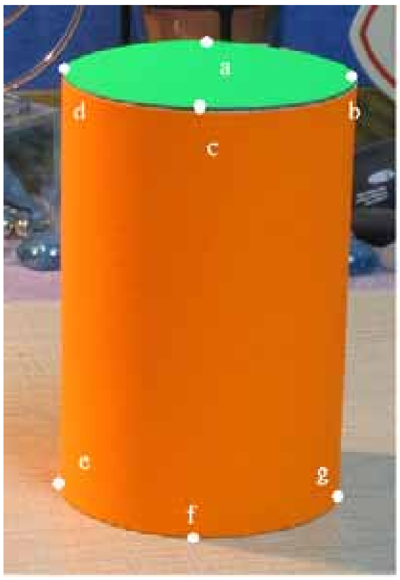
Cylindrical mapping refers to mapping a cylinder with textures. This is more difficult than mapping quadrilateral surfaces because of the curved surface, making the mapping non-linear. Three-dimensional transformations, instead of simple two-dimensional ones, are necessary. [38] [39]

16.2 Algorithm

For all kinds of texture mapping, the ultimate aim is to find a texture coordinate pair (u, v) corresponding to a given pair of image coordinates (r, c). This involves the following steps:

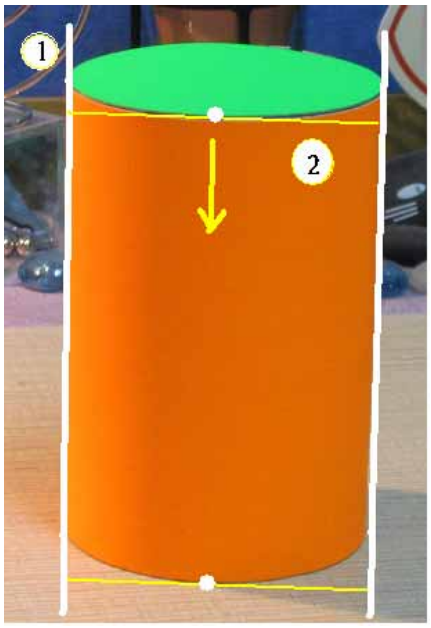
1. **Point detection**

   This involves scanning the image line by line for abrupt changes in color. The cylinder we use is covered with fluorescent paper so that any color changes can be easily detected. The points of interest are indicated in the following diagram.

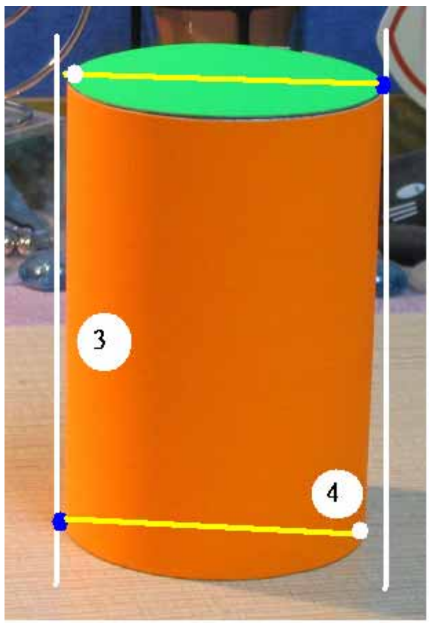


The white points are needed to define the shape of the cylinder.

Point detection, however, is not easy at all. (1) First, we detect the two edges of the cylinder using Hough Algorithm, which is described in detail in previous sections. (2) Next, the image is scanned from the top to bottom in a manner perpendicular to the two detected edges. Points a, c and f are detected with changes in color. (3) Scan the image vertically for two other corners (e and f). (4) Finally, locate the remaining two corners by simple line equations (d and g).



The white circles indicate the points found.



The blue points are found in step 3.

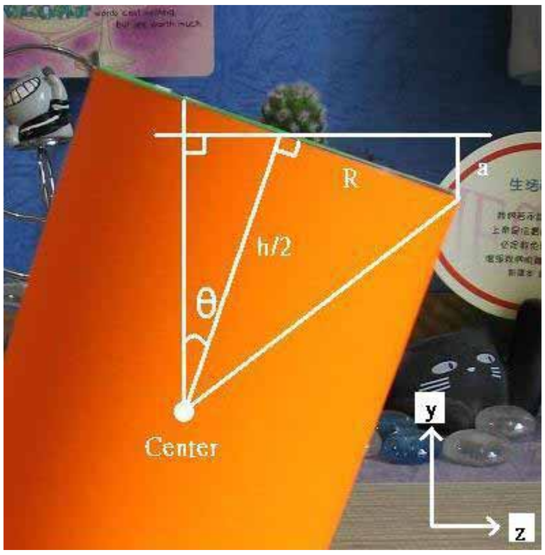
White points are found in step 4.

2. Locating the position of the cylinder center

The position of the cylinder center g as given by *(c+f)/2 + (a-c)/2*

3. Rotation along the x- and y-axes

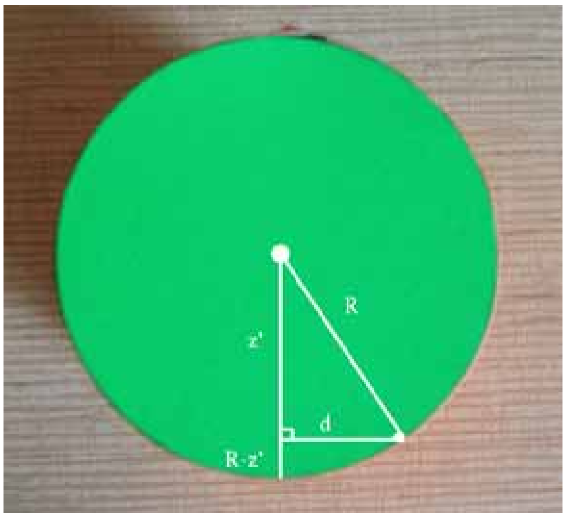
To successfully associate texture coordinates with the corresponding image coordinates, the cylinder needs to be in upright form in the view of the camera, i.e. it is aligned with the x-, y- and z-axes. First, it is rotated along the z-axis with reference to the cylinder center which is at (0, 0, 0) so that the edges of the cylinder are aligned with the y-axis. Next it is rotated along the x-axis by θ as shown below:

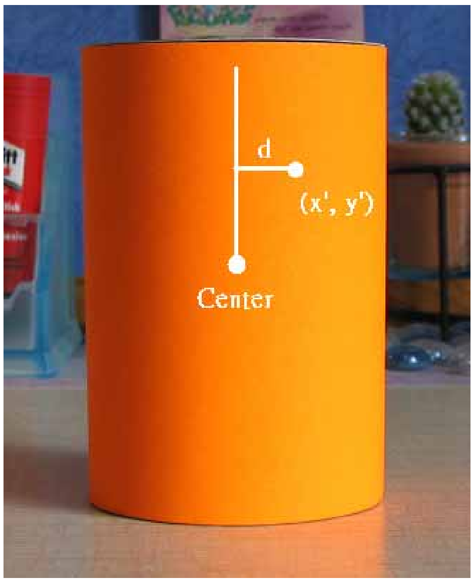


With a, R (radius of the top) and h(height) known, θ can be easily found.

4. Estimation of z-coordinate

After the transformed x' and y' coordinates are found, z' can be estimated by R – $(R^2-d^2)^{1/2}$. So, given a pair (x, y) in the (x', y', z') can be found with the cylinder center as the origin.





z' can be estimated using d which is known as the right diagram shows.

5. Texture lookup

Note that $(y'+h/2)/h = v$, where $0 \leqq v \leqq 1$ and v is the y-coordinate of texture map, $\arctan(x'/z') / 2\pi = u$.

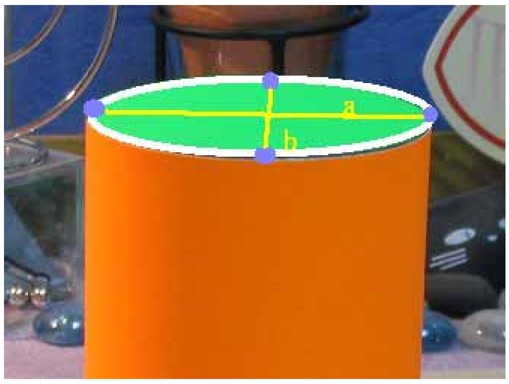
Hence, given any (x, y) in the image space, (u, v) in the texture space can be located.

16.3 Scanning the image space (cylinder)

As mentioned in previous sections, inverse mapping is better than forward mapping in that it does not need pixel sub-division calculation and anti-aliasing techniques can be used. To use inverse mapping, scan-line conversion is applied to the cylindrical surface.

1. Modeling curves with ellipse

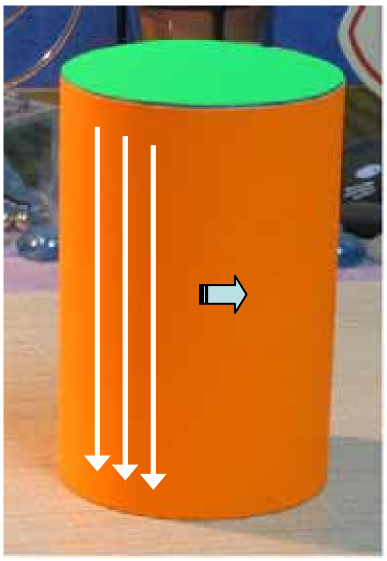
After the 4 points on the top of the cylinder are detected, they can be used to construct an ellipse in the form of $x^2/a^2 + y^2/b^2 = 1$. The calculation is complicated by slight rotation because the cylinder can be tilted.



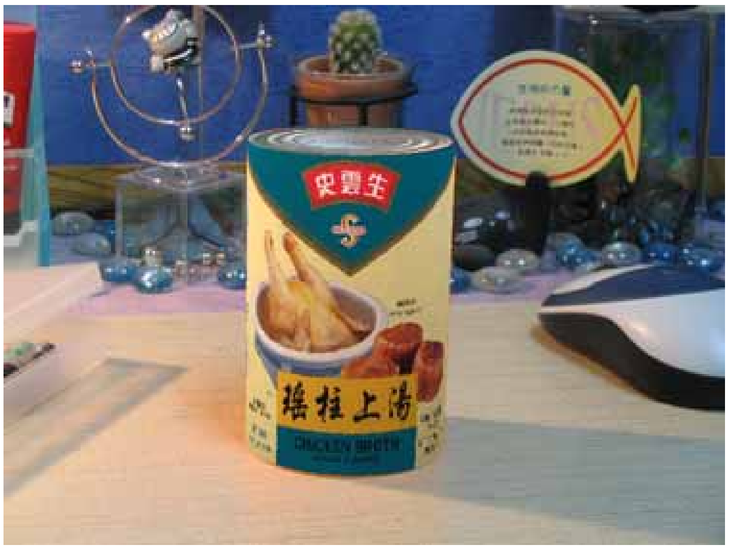
The equation of the ellipses can be estimated with the 4 detected points.

2. Scan-line conversion

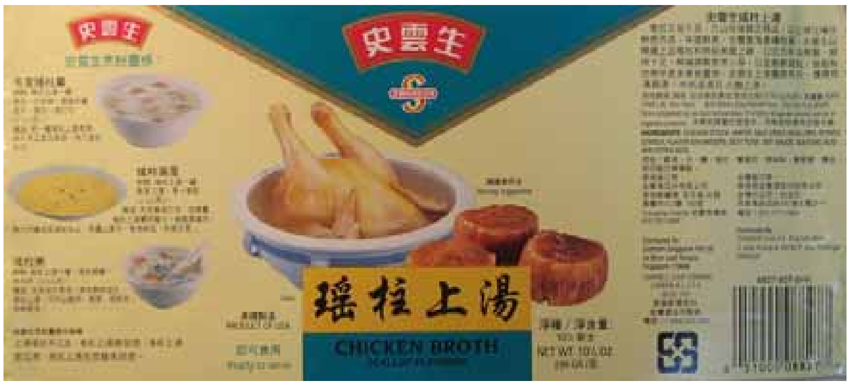
The cylinder is scan-lined from the left to the right in the following manner:



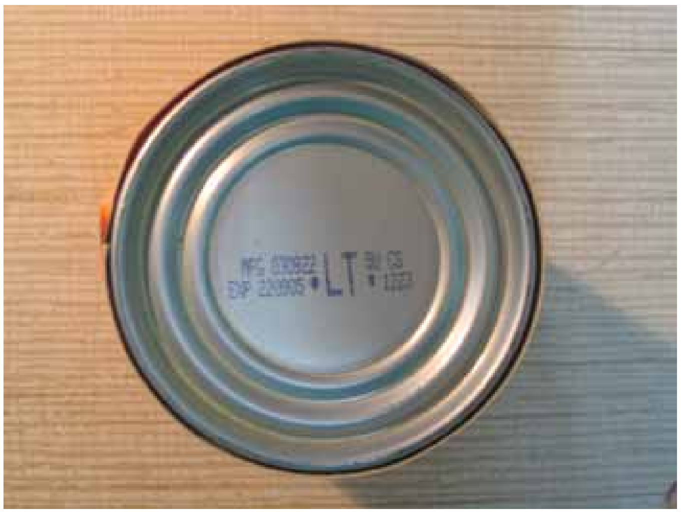
A sample output is as follows:



with texture maps :



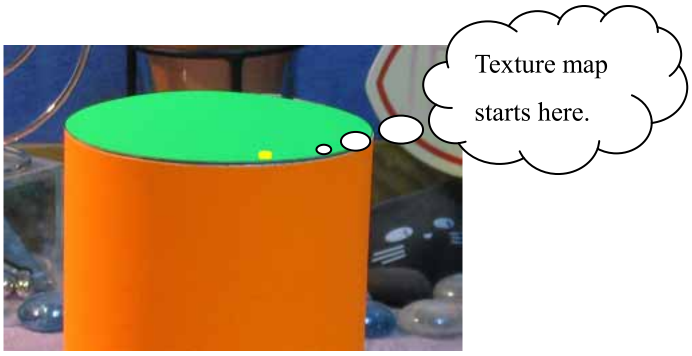
and



The top of the cylinder is also scan-line converted. However, the texture map has to be preprocessed so that the center and the radius are known.

16.4 Rotation of the cylinder

In video processing, the cylinder could be rotated. If this is not considered, the texture could stay at the same place even when the cylinder is turned around. Hence, we use a small color spot to specify on which part of the cylinder the texture map should start. The spot is located at the top near the edge, indicating that the textures (for both the top and the curved surface) start there.



The spot could be detected by data clustering, as mentioned in previous chapters. Since there are altogether 3 different markers, the colors have to be carefully chosen for accurate detection.

Chapter 17

Spherical Texture Mapping

17.1 Introduction

Spherical mapping is somewhat easier than cylindrical mapping since no matter how a sphere is oriented, it still appears as a circle in the image. This makes detection relatively easier. The texture lookup, however, is more difficult.

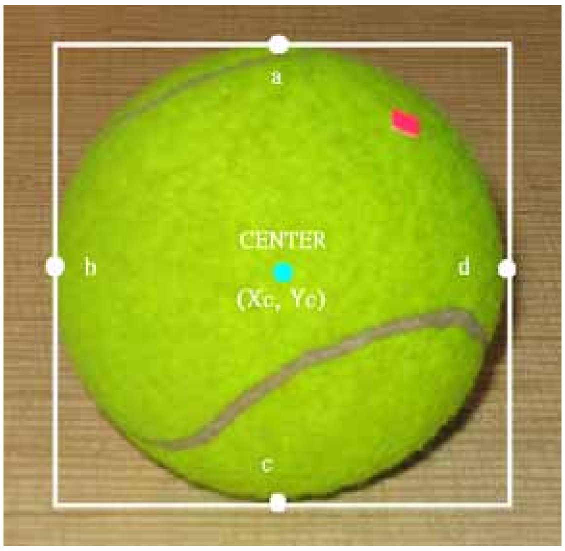
This time, a tennis ball is used as the marker because it has a fluorescent yellow color which enables more accurate detection. The pink spot on the ball indicates a "pole" which can be understood as the point where the corners of the texture map meet after mapping. There are two poles on the ball, one in pink (upper pole) and the other in orange (lower pole). [41] [42]

17.2 Algorithm

The algorithm is similar to cylindrical mapping with two parts: detection and texture lookup.

1. **Detection of points farthest away from the center, radius and center**

   This is just scanning from the four directions until a change in color is detected. The center is obtained by averaging the coordinates.



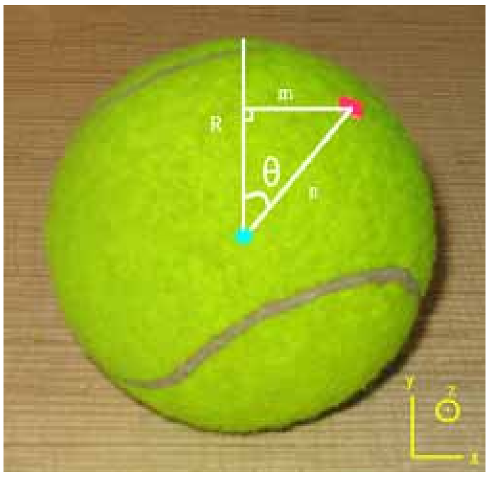
Xc = (Xb+Xd)/2 and Yc = (Ya+Yc)/2

2. **Estimation of z-coordinate**

   This is the same for the case of cylindrical mapping. (x, y, z) is obtained.

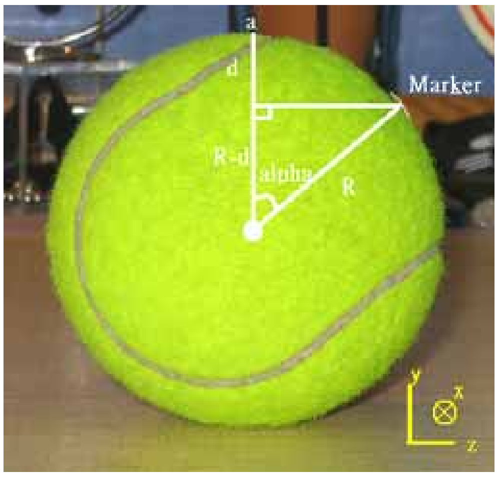
3. **Rotation along the z- and x-axes**

   Our target is to find a transformation matrix such that the pink marker will be rotated to the point *a* in the above diagram. Hence, the ball is first rotated along the z-axis by θ to make the point a, marker and the center collinear. Next the ball is rotated along the x-axis by α with reference to the center (origin) so that the marker overlaps point *a*.



(Front view) The ball is first rotated by θ

R, m and n are known.

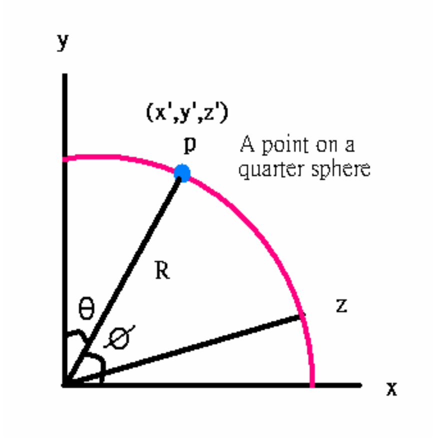


(Side view) It is then rotated by α.

R and d are used to determine α.

4. Texture lookup

So, given a pair of image coordinates (x, y), the corresponding (x', y', z') can be found by appropriate transformations and calculations. With the transformed coordinates, the texture coordinate (u, v) can be looked up according to the following relations:



$$\mathbf{x'} = r \sin(\theta) \cos(\phi)$$
$$\mathbf{z'} = r \sin(\theta) \sin(\phi)$$
$$\mathbf{y'} = r \cos(\theta)$$

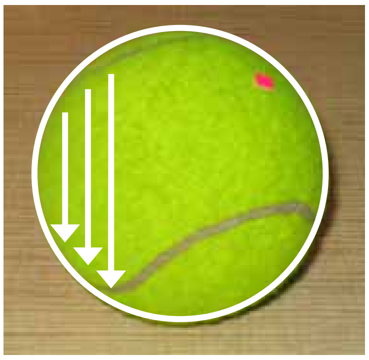
where $0 \le u, v \le 1$, and $0 \le \theta \le \pi$, $0 \le \phi \le 2\pi$

Hence, the relations become:

$$\boxed{\begin{array}{c}\mathbf{v} = \cos^{-1}(y'/R) / \pi \\ \mathbf{u} = (\cos^{-1}(x'/(R \sin(\mathbf{v}\,\pi)))\,) / (2\pi)\end{array}}$$

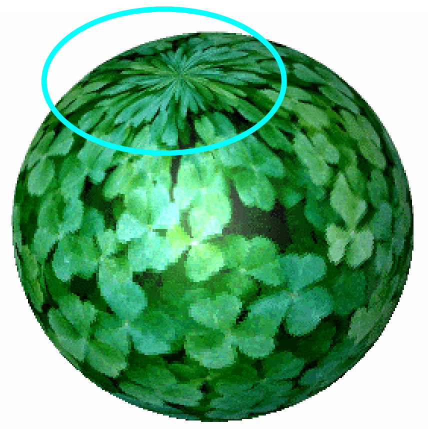
17.3 Scan-line conversion

This is to scan the circular image from top to the bottom, from left to right. The upper and lower limit of each scan-line is given by $y_c + (R^2 - (x - x_c)^2)^{1/2}$ and $y_c + (R^2 - (x - x_c)^2)^{1/2}$ respectively, for a particular x.



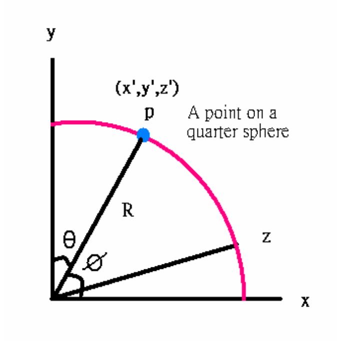
17.4 Distortion of spherical mapping

Using the above spherical mapping method, the mapped object becomes unnatural because the texture map is squeezed near the poles, as shown below.

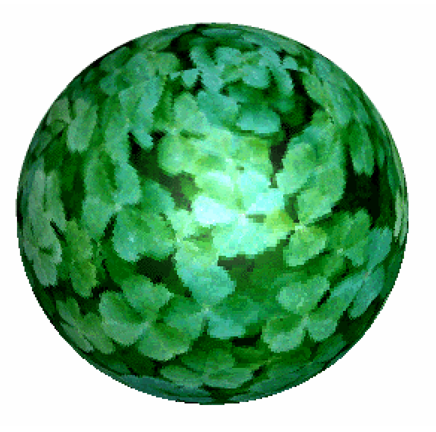


The texture is squeezed near the top

To correct this problem, the texture map can be processed so that the mapped sphere looks natural though the texture map is distorted.



Here, for a particular pixel with orientations (*θ*, *ϕ*), we replace it with (*θ*, *ϕ*sin(θ/2))* so that the texture is stretched away from the poles, as shown below:



This looks more natural after texture map distortion.

Chapter 18

Brightness Texture Mapping

18.1 Introduction

So far what we have considered is how to map a texture onto a surface of an image so that the texture map fits well with the shape of the surface, and that a user can choose what texture files to map. Those are related shape and color mapping. To make the surface more realistic, metallic or even transparent, brightness mapping should also be considered. [44]

18.2 Implementation

This is done by the conversion of RGB color space to HSV color space. After the conversion, the intensity value becomes independent of the hue value, ideal for the manipulation of shading.

18.3 Types

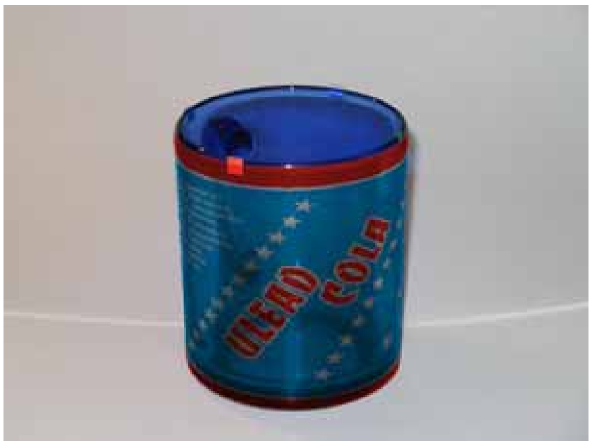
Depending on how the brightness values are manipulated, there are different kinds of shadow mapping:

1. **Direct mapping**

   This refers to mapping the texture directly without changing the brightness values. The object mapped often gets unrealistic unless the image and the texture share the same lighting environment.

2. **Mapping retaining the object brightness**

   This is not useful for markers with bright colors since the objects will acquire an abnormally bright texture after mapping. However, if the object is transparent, this will be useful in producing a translucent result.



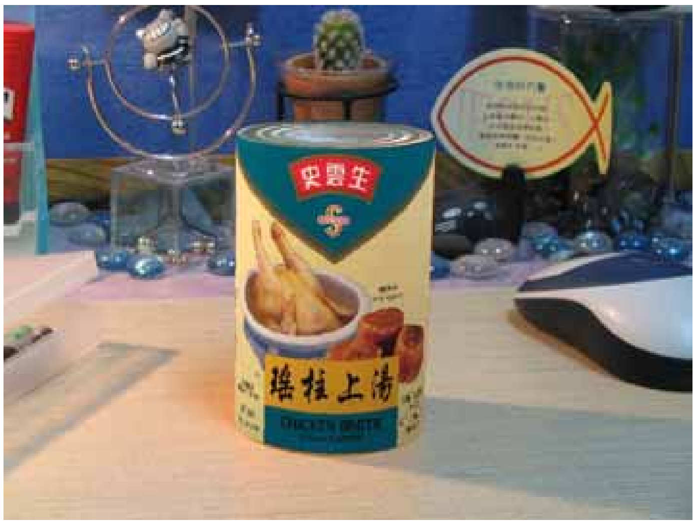
The mapped object retains to some degree of transparency.

3. Mapping using the average brightness of the surrounding

This is a very useful method in making the mapped object natural and realistic since it can “merge” with the environment with similar lighting. To further enhance reality, the brightness of the object is also considered as

(new V of a pixel) = (V of current texel) * (avg V of the surrounding) / (avg V of the object) + V of the current image pixel - (avg V of the object).

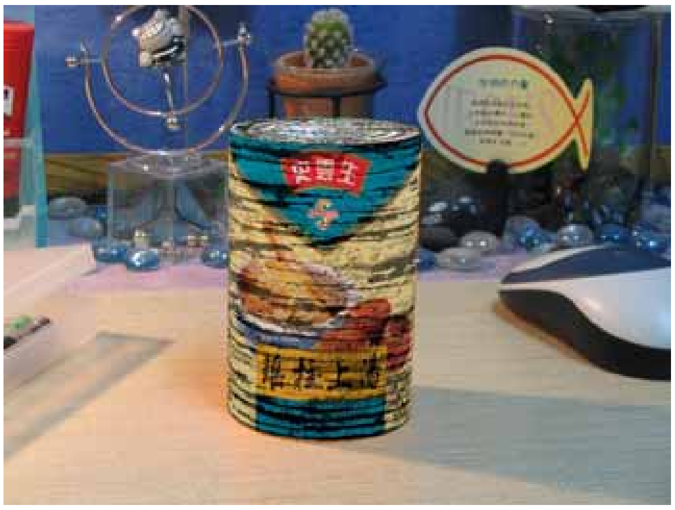
The former part is to modify the lighting based on the environment proportionally while that latter part is to add differences of the object’s shading to the mapped object, enhancing reality.



Notice the shading of the left part of the can is brighter than the right one.

4. **Mapping with a shading map**

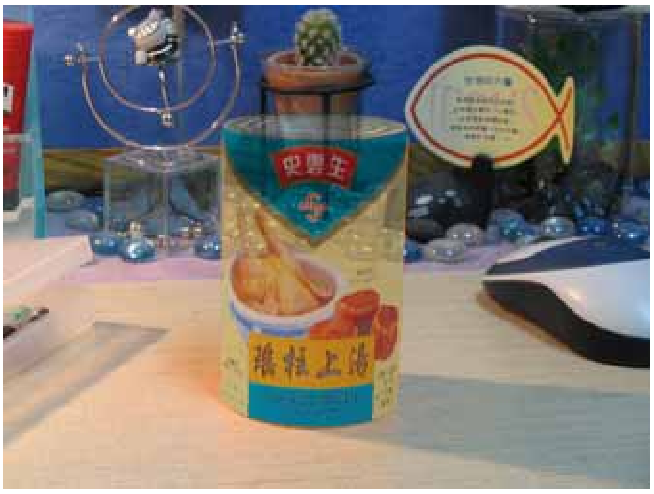
   This allows users to specify the brightness of the object with another texture map – shading map. The shading map can be any textures like grass, wood or metals. The object will then be painted with the pattern of the texture map and the brightness from another shading map.



This is a wood-like can soup.

5. **Transparency mapping**

   Not only pattern and other shading textures, but transparency can also be mapped. If we are given only one image, we have no information of what the background behind the object is like. Photographs of the environment, without the object, have to be taken. In this case, the brightness value of the object is modified as a fraction of that of the texture and that of the environment right behind it.

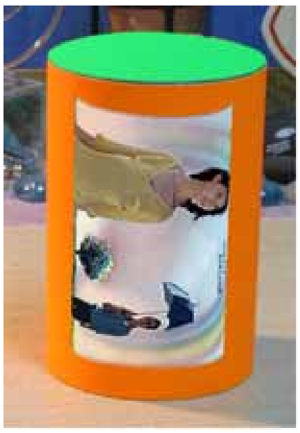


Transparency can also be mapped and the degree of transparency can be tuned.

18.4 Video mapping

Mapping different frames at different time

In a video, if the same texture map is used from the start to the end, the object only gains another texture. However, if the texture becomes different textures at different time, video mapping is possible. Video mapping can bring great interest to films.



A video is mapped to the can

Chapter 19

Performance Analysis in Part Two

19.1 Time Required to Find Out the corners of a Given Rectangular Surface

By using the new corner detection method, one can expect that the time required should be shorter than before.

	160x120	320x240	640x480	800x600
Time required for old method	< 1s	4s	10s	22s
Time required for new method	< 1s	3s	6s	13s

Figure 19.1.1 Performance comparison by using different corner detection method. The test is run on a Pentium 4 2.4GHz with 1GB of RAM.

It can be seen that the new method saves a considerable amount of time if there is nothing covering up the corner stickers.

However, if one or more corner stickers are being covered or there exists too many noise points, the program will switch back to old method to find out the corners, making the process slower than using Hough Transform statistics method directly.

	160x120	320x240	640x480	800x600
Time required for old method (s)	< 1s	4s	10s	22s
Time required for new method (s)	<1s	4s	12s	24s

Figure 19.1.2 Performance comparison by using different corner detection method when the input image has significant noise level. The test is run on a Pentium 4 2.4GHz with 1GB of RAM.

The difference in time should be small, since the process of K-Means data clustering shouldn’t take a long time for just a hundred corner sticker points.

19.2 Time Required for Texture Mapping by Using Texture Bitmaps of Different Sizes

We will now see the effect of using different size of texture images for texture mapping. The time is calculated by subtracting the corner detection time from the total image processing time.

	50x50	100x100	150x150	200x200
Time required (s)	<1s	<1s	<1s	<1s

Figure 19.2.1 Performance comparison by using different sizes of input textures. The test is run on a Pentium 4 2.4GHz with 1GB of RAM.

Since rectangular texture mapping is a very simple process, the sizes of input textures do not have an obvious impact on the performance of the image processing.

19.3 Performance of Video Frame Processing

We have used a short video clip of duration 15 seconds. Given that the frame rate is at 15 frames per second and the time required for processing the entire video is about 3.5 minutes. Then, the average processing speed is about one frame per second, which is faster than it was in the last semester. Unfortunately, this is still not enough for real-time video processing.

Chapter

20

Conclusion for the final year project

At this stage, a simple video object tracking and texture mapping system is finished. With its user-friendly graphical user interface, users can input movies with an object which is a cube, a cylinder or a sphere. By object detection, the objects can be identified and mapped with different patterns, shades or even transparency. This application is very useful in video production in which advertisements are allowed and in computer games where reality is necessary.

Throughout the project, we have learned but not limited to the following:

1. MFC MDI programming skill
2. DirectShow programming skill
3. The technique of scan-line conversion
4. Basic digital image processing skills
5. 3-D geometrical transformations

There are many obstacles in the development process but we were able to solve them systematically. We express our cordial thanks to Prof. Lyu and the researchers how helped us in the project.

Reference

[1] About.com, Kathleen McGinn, November 14, 2001
http://inventors.about.com/gi/dynamic/offsite.htm?site=http://www.princetoninfo.com/200111/11114c01.html

[2] 70th Oscar Academy Awards, 1997.
http://www.oscar.com/legacy/pastwin/visual_eff1.html

[3] KillerMovies.com, Community Forums.
http://www.killermovies.com/forums/archive/index.php/t-15731

[4] DVDRHelp.com, What is VCD?
http://www.dvdrhelp.com/vcd

[5] James D. Murray & William vanRyper, "Encyclopedia of Graphic File Formats", O'Reilly & Associates, Inc.

[6] Nancy L. Price, "Graphic File Format Comparison"
http://www.uwm.edu/People/price/graphic_file_format_comparison.html

[7] Christopher C. Yang, "Color Coordinate Systems"
http://www.se.cuhk.edu.hk/~yang/research/CIP/color/ch2.ps

[8] TutorGig Encyclopedia, "HSV Color Space"
http://www.tutorgig.com/encyclopedia/getdefn.jsp?keywords=HSV_color_space

[9] Coder.com, "Using Edge Detection"
http://coder.com/creations/banner/examples/edge.html

[10] HyperMedia Image Processing Reference, "Gaussian Smothing"
http://www.cee.hw.ac.uk/hipr/html/gsmooth.html

[11] HyperMedia Image Processing Reference, "Mean Filter"
http://www.cee.hw.ac.uk/hipr/html/mean.html

[12] HyperMedia Image Processing Reference, "Median Filter"
http://www.cee.hw.ac.uk/hipr/html/median.html

[13] Jiqiang Song, Min Cai, Michael R. Lyu and ShiJie Cai, "A New Approach for Line Recognition in Large-size Images Using Hough Transform"

[14] School of Computing Science at SFU, "Basics of Video"
http://www.cs.sfu.ca/CourseCentral/365/li/material/notes/Chap3/Chap3.4/Chap3.4.html

[15] Microsoft Foundation Class reference, Microsoft Corporation
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcmfc98/html/_mfc_Class_Library_Reference_Introduction.asp

[16] Multiple Document Interface sample, Microsoft Corporation
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcsample/html/_sample_mfc_SCRIBBLE.asp

[17] MPEG Starting Points and FAQs,
http://www.mpeg.org/MPEG/starting-points.html

[18] International Telecommunication Union,
http://www.itu.int/home/

[19] Introduction to DirectShow, Microsoft Corporation,
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directX/htm/introductiontodirectshow.asp?frame=true

[20] DirectX 9.0 SDK Update (Summer 2003) C++, Microsoft Corporation,
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/directx9cpp.asp

[21] About DirectShow Filters, Microsoft Corporation,
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directX/htm/aboutdirectshowfilters.asp

[22] The Filtergraph and Its Components, Microsoft Corporation,
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directX/htm/overviewofdataflowindirectshow.asp

[23] DivX.com,
http://www.divx.com/

[24] XviD.org,
http://www.xvid.org/index-static.html

[25] K-Means Clustering Algorithm,
http://cne.gmu.edu/modules/dau/stat/clustgalgs/clust5_bdy.html

The following references are also helpful in the project development stage:

[26] United Nations Educational, Scientific and Cultural Organization, "Basic Edge Detection"
http://www.netnam.vn/unescocourse/computervision/42.htm

[27] HyperMedia Image Processing Reference, "Canny Edge Detector"
http://www.cee.hw.ac.uk/hipr/html/canny.html

[28] HyperMedia Image Processing Reference, "Roberts Cross Edge Detector"
http://www.cee.hw.ac.uk/hipr/html/roberts.html

[29] HyperMedia Image Processing Reference, "Sobel Edge Detector"
http://www.cee.hw.ac.uk/hipr/html/sobel.html

[30] Texture Mapping as a Fundamental Drawing Primitive - Paul Haeberli and Mark Segal - June 1993
http://www.sgi.com/grafica/texmap/

[31] WinTexture - A New Design Application from De Montfort University
http://www.staff.dmu.ac.uk/~gfo/wtx.html

[32] Rochester Institute of Technology - Computer Graphics II
http://www.cs.rit.edu/~ncs/Courses/571/syllabus.shtml

[33] Computer Graphics C Version 2nd Edition - Donald Hearn, M. Pauline Baker

[34] University of Toronto - Department of Computer Science
CSC320F: Introduction to Visual Computing
http://www.cs.toronto.edu/~kyros/courses/320/Lectures/lecture-15.pdf

[35] Cornell University - Introduction to Computer Graphics - Spring 2003
http://www.cs.cornell.edu/Courses/cs417/2003sp/Lectures/Lecture25/25texture.pdf

[36] Forward Image Mapping - Baoquan Chen, Frank Dachille and Arie Kaufman
http://www-users.cs.umn.edu/~baoquan/papers/fim.pdf

[37] Model-Based Rendering - Bob Fisher 2003
http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/LIVATINO2/MainApprRVVS/node2.html

[38] Cylindrical Texture Mapping –
http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/textureMapping/cylindrical_texture_mapping_guide.html

[39] Unigraphics Render Tricks and Tips
http://www.vickers.de/ug_render/uv_texture_mapping.htm

[40] Texture Mapping
http://astronomy.swin.edu.au/~pbourke/texture/texturemapping/

[41] Teaching Texture Mapping Visually
http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm

[42] Spherical Textures
http://www.vterrain.org/Textures/spherical.html

[43] Texture map correction for spherical mapping
http://astronomy.swin.edu.au/~pbourke/texture/polargrid/

[44] NuGraf Toolkit Tutorial
http://www.okino.com/new/toolkit/tut/main10.htm

[45] The Code Project,
http://www.codeproject.com

Work division

Tong Pak Hin: Ch 1-6, 9-15, 19
Yuen Pak Kei: Ch 7-9, 10, 16-18, 19-20

Acknowledgement

To

Dr. Michael R. Lyu

Mr. Edward H. Yau

Mr. Song Ji Qiang

for providing us with valuable comments and suggestions
in the development of this final year project

Appendix

Workable Components Public Function Overview

Bitmap File Reader (BMPIo.h)

- unsigned char*** readBMP(char* filename)
  This reads the pixel 2D array from file "filename".
- void writeBMP(unsigned char*** pixs, char* outName)
  This writes a pixel array "pixs" to a bitmap file "outName".
- int getW()
  This returns the width of the bitmap file.
- int getH()
  This returns the height of the bitmap file.

RGB/HSV Converter (HSVCvrt.h)

- unsigned char* RGBtoHSV(unsigned char* pixs)
  This converts an RGB 3-byte block to an HSV one.
- unsigned char* HSVtoRGB(unsigned char* pixs)
  This does the reverse of the above function.

Edge Detector (EdgeFind.h)

- void process(int temp, int temp1, unsigned char*** pixs)
  This picks out edge points using "temp" as the hue, "temp1" as the saturation value from the pixel array "pixs".
- int** getPts()
  This returns the edge point list.
- int getSize()
  This returns the number of edge points found.

Linear Equation Finder (HTrans.h)

- int** getList(int** pts, int size)
  This automatically tries an optimal tolerance value.

- int* process(int** pts, int size, int tol)
  This derives equations from "size" edge points "pts" with tolerance value "tol".
- int getSize()
  This returns the size of the equation list.

✚ Equation Processor (EqnPro.h) (This class has been deleted in the second semester)

- int** process(int** list, int lSize)
  This tries to eliminate redundant equations from list "list" with size "lSize".
- int getSize()
  This returns the size of equation list after processing.
- int solveX(int* eqn1, int* eqn2)
  This solves for the value of x from two equations "eqn1" and "eqn2".
- int solveY(int* eqn1, int x)
  This solves for the value of y from an equation "eqn1" and a given x.

✚ Median Filter Smoother (Smooth.h)

- unsigned char* process(unsigned char* pixs)
  This returns a smoothed version of the pixel array "pixs".

✚ Texture mapper (TexMap.h)
- char *TexMap::process(char *pixin)
  This processes the texture map pixin and maps it to the image file with the location specified by four vertices.

✚ KMeans calculator (KMeans.h)
- char *KMeans::process(char *pixin)
  This finds out the group centroids of the corner point groups.

✚ Cylindrical texture mapper (CyMap.h)
- char *CyMap::process(char *pixin)
  This maps texture onto a cylinder.

✚ Spherical texture mapper (SpMap.h)
- char *SpMap::process(char *pixin)
  This maps texture onto a sphere.