UrduKashishStyler

Developed By:

Khurram Iqbal

Muhammad Ali

Supervised By:

Dr. Sikander Hayat Khiyal

Department of Computer Science
International Islamic University Islamabad
(2003)

Department of Computer Science

International Islamic University Islamabad

Final Approval Dated December 26, 2003

It is certified that we have read the project report submitted by Mr. Khurram Iqbal Reg No14-CS/MS/01 and Mr. Muhammad Ali Reg No 17-CS/MS/01, it is our judgment that this report is of sufficient standard to warrant to acceptance by International Islamic University Islamabad for Master of Science Degree in Computer Sciences.

Committee

1. External Examiner
   Dr. Muhammad Afzal
   Director CIT, Arid Agriculture
   University, Muree Road, Rawalpindi.

2. Internal Examiner
   Mr. Zaheer Aziz
   Assistant Professor
   Department of Computer Sciences
   Faculty of Management Sciences

3. Supervisor
   Dr. Sikander Hayat Khiyal
   Head Department of Computer Sciences
   Faculty of Management Sciences
   International Islamic University Islamabad.

A dissertation submitted to the International Islamic University, Islamabad as a partial fulfillment of the requirement for the award of the degree of Master of Science in Computer Science.

Dedication

To our parents

Declaration

We here by solemnly declare that the developed software and this accompanied report neither as a whole nor as a part thereof has been copied from any source. It is further declared that we have developed this software as well as the accompanied report entirely on the basis of our personal efforts made under the kind guidance of our teachers. If any part of this thesis is proved to be copied out from any source or found to be reproduction of some other, we shall stand by the consequences thereof. We also declare that no part or whole of the work presented in this thesis has been submitted in support of any degree for other university or institution of learning.

Khurram Iqbal
Reg # 14-CS/MS/01

Muhammad Ali
Reg # 17-CS/MS/01

Acknowledgement

First and above all, gratitude is due to ALLAH who gave us health, strength and patience to complete this thesis. We are thankful to our teachers who guided us in this project in any way they could, specially to our supervisor Dr. Sikander Hayat Khiyal for providing us help in conducting the project.

We further wish to express our gratitude to people we met during project who have contributed to this thesis by offering guidance, sharing good advice, and providing tough critique when necessary.

Khurram Iqbal
Reg # 14-CS/MS/01

Muhammad Ali
Reg # 17-CS/MS/01

Project In Brief

Project Title:	UrduKashishStyler
Undertaken By:	Khurram Iqbal 14-CS/MS/01 Muhammad Ali 17-CS/MS/01
Supervised By:	Dr. Sikander Hayat Khiyal
Date:	November 6, 2003
Start Date:	September, 2002
End Date:	July, 2003
Tools:	Visual C++
Operating System:	Windows XP

Abstract

UrduKashishStyler is a vector-based tool specially designed for Urdu text designing to ease and reduce the text designing efforts, because text composing with effects in existing tools is a tedious job and require more than fifteen steps. With the existing systems only skilled person can prepare good looking headings. This tool provides a number of collections with different composing styles and patterns in *Kashish Art Gallery*, from where the user can select the sample and the pattern of selected sample will be applied on the text. The user may also customize these patterns.

We have used true type vector fonts for writing Urdu. In vector fonts the shape of glyphs is stored in form of vectors. After obtaining data from font file we decipher the information into vector points by using deCasteljau algorithm. Then these vector points are used to draw the actual glyphs and performing other operation like gradient-filling, pattern filling and transformations.

Preface

This thesis is regarding the project work title "UrduKashishStyler" submitted as partial fulfillment of requirement for the award of the Master of Science Degree in Computer Sciences International Islamic University, Islamabad.

In chapter one, we present the introduction of the project, giving the purpose and need of the Project. Objectives of the project are also mentioned in this chapter.

In chapter two, we present general background of fonts and technologies used in details. Chapter three is concerned with system analysis, which cover requirement and domain analysis of this software with the help of different software models. Chapter four deal with the design analysis describing the design of the software. In Chapter five, we give the implementation details of our software along with some sample codes and techniques that are used in the project. In Chapter six, we define Testing and Evaluation part showing the system stability.

At the end of Dissertation Appendices, Glossary and References are given. In the appendix-A a general description of software is given.

Table of Contents

Figures Used in Project Documentation

Figure No. **Page No.**

Chapter 1

Introduction

1. Introduction

Urdu, a language full of beauty and grace, a language that seems to have been custom-built for literature, a language that adds meaning to prose and charm to poetry, a language that is spoken in many countries of the world. To present Urdu with its grace and beauty in the newspapers, magazines and business advertisements composing and designing tools play an important role.

1.1 Vector Image

Vector images, also called object-oriented images, are created through a sequence of commands or mathematical statements that place lines and shapes in two-dimensional system. In physics, a vector is a representation of both quantity and direction at the same time. In vector graphics, the file that results from a graphic artist's work is created and saved as a sequence of vector statements. For example, instead of containing a bit in the file for each bit of a line drawing, a vector graphic file describes a series of points to be connected. One result is a much smaller file. Graphical elements in a vector file are called objects, where each object is a self-contained entity, having properties such as color, shape, outline, size, and position on the screen included in its definition. Since each object is a self-contained entity, we can move and change its properties over and over again while maintaining its original clarity without affecting other objects in the drawing. These characteristics make vector-based applications ideal for text designing, where the design process often requires individual objects to be created and manipulated. Vector-based text designing is resolution independent. This means that they appear at the maximum resolution of the output device, such as your printer or monitor. As a result, the image quality of drawing is a higher quality resolution.

1.2 Glyph and Vector Glyph

Glyph is the representation of a character. In Urdu many of the characters can have more than one glyphs i.e. Isolate, initial, middle and final shown in figure 1.1. The words are constituted with different combination of these glyphs provided that initial form always

comes first, final in the last and the middle is positioned between the first and last form of glyphs and may vary in numbers.

Figure 1.1 Isolate, final, middle and initial glyphs

Vector Glyph is the glyph in the form of outlines. The *Outline* of glyph is a series of intermingled polyline and qspline records. The deCasteljau algorithm is applied on these records to get a series of points that are used to make the glyphs in vector form as shown in figure 1.2. As vector glyph is resolution independent so when reducing or increasing the size, it remains same without the loss of quality. It is the characteristic which makes vector-based applications ideal for text designing.

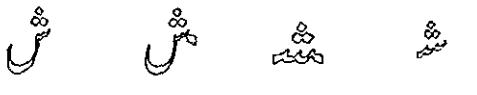


Figure 1.2 Isolate, final, middle and initial vector glyphs

The greatest thing about storing characters as outline is that only one outline per character is needed to produce all the sizes of that character. A single outline can be scaled to an enormous range of different sizes. It enables the same character to be displayed on monitors of different resolutions, and to be printed out at a large number of different sizes.

1.3 Existing Problem

There are different composing and designing tools available for different languages, which play an important role for better presentation of the text written in simple format. When the vector form of glyphs are joined to make composite vector glyphs, then it becomes very essential in various applications such as text designing in newspapers, magazines etc. The existing applications that allow writing Urdu text in vector form do not cope with the degenerate cases, such that the resultant composite vector glyph has the intersecting line,

common in the individual glyphs. Let us take a brief survey of existing applications that allow writing Urdu vector glyphs.

Figure 1.3 Vector glyphs for Urdu word KASHISH

The existing applications write these individual glyphs in composite form as follows

Figure 1.4 Composite vector glyphs for Urdu word KASHISH

The problem with the above composite vector glyph is that there are joining lines in between each of the individual vector glyph, that makes each of the glyph as a separate entity. Another problem in the existing tools is that the composing and designing of Urdu text with effects is very difficult and requires more than fifteen steps that are possible only by a skilled person.

1.4 Objective

To overcome the problem mentioned above, our objective is to develop an algorithm that takes the individual vector glyph as input and the resultant output is the composite vector glyph with no intersecting line that was common between the individual glyphs. We also want to develop a software application which provides the facility to develop good looking heading with few mouse clicks that is possible in existing systems with number of steps. For this purpose a number of collections with different composing styles and patterns will be available in *Kashish Art Gallery* (Figure 1.5), from where the user can select the sample and the pattern of selected sample will be applied on the text.

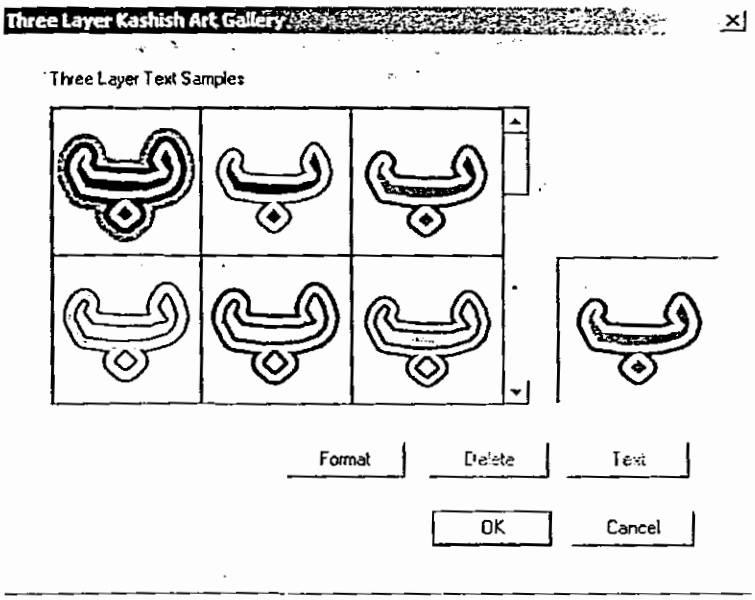


Figure 1.5 Kashish Art Gallery with different samples.

Chapter 2

Basic Concepts

2. Basic Concepts

To present Urdu with its grace and beauty in the newspapers, magazines and business advertisements, designing applications play an important role. These applications are based on creating either vector or bitmap images. UrduKashishStyler is a comprehensive vector-based drawing program that makes it easy to create professional artwork for intricate Urdu text designing technical illustrations. UrduKashishStyler provide tools that work efficiently producing high-quality text designing.

2.1 Fonts

The text is displayed using a Font that refers to a complete set of glyphs in a specific typeface, style and weight. UrduKashishStyler uses the TrueType Fonts (TTF) instead of Bitmap Fonts (BF) since Bitmap Fonts need separate bitmap sets for each and every font size whereas TrueType Fonts remain essentially the same regardless of the size of a character. TTF and BF are explained in the following section.

2.1.1 TrueType Font

A TrueType is a scalar of scalable font. A TTF is defined using mathematical vectors, so it remains essentially the same regardless of the size of a character. As a result range of sizes can be rendered from the same definition. Moreover lines and curves instead of pixels are used by TTF for drawing glyphs. These lines and curves have no designated point size.

TrueType font technology is designed by Apple Computer, and now used by both Apple and Microsoft in their operating systems. Microsoft has distributed millions of quality TrueType fonts in hundreds of different styles, including them in its range of products and the popular TrueType Font Packs. TrueType fonts offer the highest possible quality on computer screens and printers, and include a range of features which make them easy to use.

The TrueType font technology consists of two components: the TrueType font files and TrueType rasterizer.

2.1.1.1 TrueType Font Files

A TrueType font file includes different kind of information used by the operating system software to ensure that characters are displayed on the computer screen or are printed out exactly as the font designer intended them to be. The information in a TrueType font is arranged in a series of tables. In addition to the shapes of each character, a TrueType font file includes information about how the characters should be spaced within a block of text, character mapping details (governing the variety of characters included in the font and the keystrokes needed to access them), and much more besides.

2.1.1.2 TrueType Rasterizer

The TrueType Rasterizer is a computer program which is typically incorporated as a part of an operating system or printer control software. With this in mind, it has been written with a well defined client interface, and a clean modular structure in portable C.

The job of the TrueType Rasterizer is to generate character bitmaps for screens and printers (otherwise known as raster devices). It accomplishes this by performing the following tasks:

- Reading the outline description of the character (lines and splines) from the TrueType font file.
- Scaling the outline description of the character to the requested size and device resolution.
- Adjusting the outline description to the pixel grid (based on hinting information).
- Filling the adjusted outline with pixels (scan conversion).

2.1.2 Bitmap Font

A bitmap font represents each character glyph using a bitmap array and is designed for a specific aspect ratio and character size. Since the logical size of the bitmap is fixed, its physical size on the display device will depend upon the resolution of the device. Either the bitmap fonts need separate bitmap sets for each and every font size, or larger character sizes

are created by simply duplicating rows or columns of pixels. However this can be done in integral multiples only and with certain limits. For this reason bitmap fonts are termed non-scalable fonts. They can not be extended or compressed to an arbitrary size.

2.2 GDI + (Graphics Device Interface)

Microsoft Windows GDI+ is a class-based application programming interface (API) for C/C++ programmers. It enables applications to use graphics and formatted text on both the video display and the printer.

A graphics device interface, such as GDI+, allows application programmers to display information on a screen or printer without having to be concerned about the details of a particular display device. The application programmer makes calls to methods provided by GDI+ classes and those methods in turn make the appropriate calls to specific device drivers. GDI+ insulates the application from the graphics hardware, and it is this insulation that allows developers to create device-independent applications.

As its name suggests, GDI+ is the successor to Windows Graphics Device Interface (GDI), the graphics device interface included with earlier versions of Windows. Windows XP or Windows Server 2003 supports GDI for compatibility with existing applications. GDI+ optimizes many of the capabilities of GDI and provides additional features as well.

2.2.1 Parts of GDI+

The services of Microsoft Windows GDI+ fall into three broad categories: 2-D vector graphics, Imaging and Typography

2.2.1.1 2-D Vector Graphics

Vector graphics involves drawing primitives (such as lines, curves, and figures) specified by sets of points on a coordinate system. For example, a straight line can be specified by its two endpoints, and a rectangle can be specified by a point giving the location of its upper-left corner and a pair of numbers giving its width and height. A simple path can be specified by an array of points to be connected by straight lines. A Bézier spline is a sophisticated curve

specified by four control points. GDI+ provides classes that store information about the primitives themselves, classes that store information about how the primitives are to be drawn, and classes that actually do the drawing.

2.2.1.2 Imaging

Certain kind of pictures is difficult or impossible to display with the techniques of vector graphics. For example, the pictures on toolbar buttons and the pictures that appear as icons would be difficult to specify as collections of lines and curves. A high-resolution digital photograph of a crowded baseball stadium would be even more difficult to create with vector techniques. Images of this type are stored as bitmaps, arrays of numbers that represent the colors of individual dots on the screen. Data structures that store information about bitmaps tend to be more complex than those required for vector graphics, so there are several classes in GDI+ devoted to this purpose.

2.2.1.3 Typography

Typography is concerned with the display of text in a variety of fonts, sizes, and styles. GDI+ provides an impressive amount of support for this complex task. One of the newest features in GDI+ is sub pixel antialiasing, which gives text rendered on an LCD screen a smoother appearance.

2.2.2 Features of GDI+

The GDI+ has several features that allow application programmers to display information on a screen or printer without being concerned about the details of a particular display device. Following section describe these features.

2.2.2.1 Drawing Tools

GDI+ provides a variety of drawing tools to use in device contexts. It provides pens to draw lines, brushes to fill interiors, and fonts to draw text.

2.2.2.2 Gradient Brushes

GDI+ expands on Windows Graphics Device Interface (GDI) by providing linear gradient and path gradient brushes for filling shapes, paths, and regions. Gradient brushes can also be used to draw lines, curves, and paths. When a shape is filled with a linear gradient brush, the color gradually changes as it moves across the shape. For example, suppose a horizontal gradient brush is created by specifying blue at the left edge of a shape and green at the right edge. When the shape is filled with the horizontal gradient brush, it will gradually change from blue to green as it moves from its left edge to the right edge. Similarly, a shape filled with a vertical gradient brush will change color as moved from its top to bottom. The following illustration shows an ellipse filled with a horizontal gradient brush and a region filled with a diagonal gradient brush shown in figure 2.5.

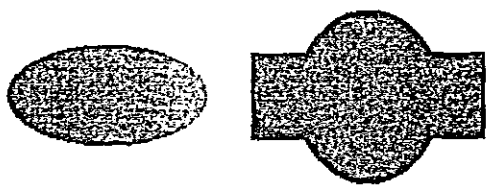


Figure 2.5 Shape filled by gradient fill

When a shape is filled with a path gradient brush, then there are a variety of options for specifying how the colors change as it moves from one portion of the shape to another. One option is to have a center color and a boundary color so that the pixels change gradually from one color to the other as it moves from the middle of the shape towards the outer edges.

2.2.2.3 Cardinal Splines

GDI+ supports cardinal splines, which are not supported in GDI. A cardinal spline is a sequence of individual curves joined to form a larger curve. The spline is specified by an array of points and passes through each point in that array. A cardinal spline passes smoothly (no sharp corners) through each point in the array and thus is more refined than a path created by connecting straight lines. Figure 2.6 and 2.7 shows two paths, one created by connecting straight lines and one created as a cardinal spline.

Figure 2.6 Path created by cardinal spline

Figure 2.7 Path created by connecting straight lines

2.2.2.4 Independent Paths

GDI+ provides drawing facility using a Graphics object. Several GraphicsPath objects are created and destroyed from the Graphics object. A GraphicsPath object is not destroyed by the drawing action, so the same GraphicsPath objects can be used to draw a path several times.

2.2.2.5 Transformations and the Matrix

GDI+ provides the Matrix facility, a powerful tool that makes transformations (rotations, translations, and so on) easy and flexible. A single 3×3 matrix can store one transformation or a sequence of transformations. Figure 2.8 shows a path before and after a sequence of two transformations (first scale and then rotate).

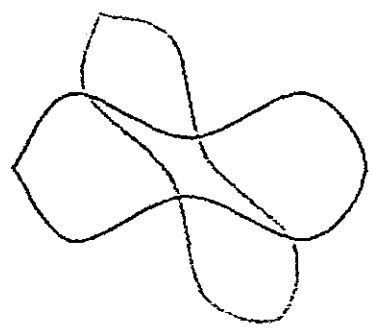


Figure 2.8 Path with two transformations (scale and rotate)

2.2.2.6 Scalable Regions

GDI+ expands greatly on GDI with its support for regions. In GDI, regions are stored in device coordinates, and the only transformation that can be applied to a region is a translation. GDI+ stores regions in world coordinates and allows a region to undergo any transformation (scaling, for example) that can be stored in a transformation matrix. Figure 2.9 shows a region before and after a sequence of three transformations: scale, rotate, and translate.

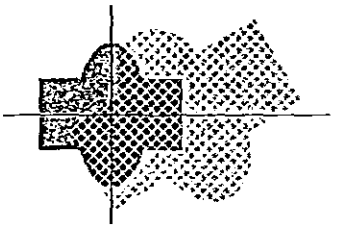


Figure 2.9 Region with scale, rotate and translate

2.2.2.7 Alpha Blending

In figure 2.9, there is the untransformed region (filled with red) through the transformed region (filled with a hatch brush). This is made possible by alpha blending, which is supported by GDI+. With alpha blending, the transparency of a filled color can be specified. A transparent color is blended with the background color; the more transparent the fill color is made, the more the background shows through. Four ellipses that are filled with the same color (red) at different transparency levels have been shown in figure 2.15 below.

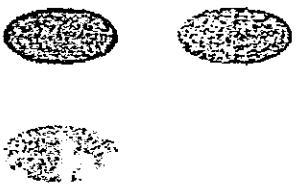


Figure 2.10 Ellipse with different transparency levels

2.2.2.8 Support for Multiple Image Formats

GDI+ provides the Image, Bitmap, and Metafile classes, that allows loading, saving and manipulating images in a variety of formats. The following formats are supported: BMP, Graphics Interchange Format (GIF), JPEG, PNG, TIFF, ICON, WMF, EMF.

2.3 Device Context

Device independence is one of the chief features of Microsoft Windows. Applications can draw and print output on a variety of devices. The software that supports this device independence is contained in two dynamic-link libraries. The first, Gdiplus.dll, is referred to as the GDI+, the second is referred to as a device driver. The name of the second depends on the device where the application draws output. For example, if the application draws output in the client area of its window on a VGA display, this library is Vga.dll; if the application prints output on an Epson FX-80 printer, this library is Epson9.dll.

An application must inform GDI+ to load a particular device driver and, once the driver is loaded, to prepare the device for drawing operations (such as selecting a line color and width, a brush pattern and color, a font typeface, a clipping region, and so on). These tasks are accomplished by creating and maintaining a device context (DC). A device context is a structure that defines a set of graphic objects and their associated attributes, and the graphic modes that affect output. The graphic objects include a pen for line drawing, a brush for painting and filling, a bitmap for copying or scrolling parts of the screen, a palette for defining the set of available colors, a region for clipping and other operations, and a path for painting and drawing operations. Unlike most of the structures, an application never has direct access to the device context; instead, it operates on the structure indirectly by calling various functions.

2.3.1 Device Context Types

There are four types of device context (DC): display, printer, memory (or compatible), and information, where each type serves a specific purpose.

2.3.1.1 Display

It supports drawing operations on a video display. The application obtains a display device context to identify the window in which the corresponding output will appear. Whenever an application needs to draw in the client area then it obtains a display device context that is released when the drawing is finished. There are three types of Device context for video displays:

1. **Class Device Contexts** are supported strictly for compatibility with 16-bit versions of Windows. When writing the applications, avoid using the class device context; use a private device context instead.

2. **Common Device Contexts** are display device context maintained in a special cache by the system. Common device contexts are used in applications that perform infrequent drawing operations. Before the system returns the device context handle, it initializes the common device context with default objects, attributes, and modes. Any drawing operations performed by the application use these defaults unless one of the GDI+ functions is called to select a new object, change the attributes of an existing object, or select a new mode. Because only a limited number of common device contexts exist, an application should release them after it has finished drawing. When the application releases a common device context, any changes to the default data are lost.

3. **Private Device Contexts** are display device context that, unlike common device contexts, retain any changes to the default data, even after an application releases them. Private device contexts are used in applications that perform numerous drawing operations such as computer-aided design (CAD) applications, desktop-publishing applications, drawing and painting applications, and so on. Private device contexts are not part of the system cache and therefore need not be released after use. The system automatically removes a private device context after the last window of that class has been destroyed.

2.3.1.2 Printer

The printer device context supports printing on a dot-matrix printer, ink-jet printer, laser printer, or plotter. An application creates a printer device context by supplying the appropriate arguments (the name of the printer driver, the name of the printer, the file or device name for the physical output medium, and other initialization data) and after the application has finished printing, it deletes the printer device context.

2.3.1.3 Memory

To enable applications to place output in memory rather than sending it to an actual device, a special device context is used for bitmap operations called a *memory device context.* A memory device context enables the system to treat a portion of memory as a virtual device. It is an array of bits in memory that an application can use temporarily to store the color data for bitmaps created on a normal drawing surface. Because the bitmap is compatible with the device, a memory device context is also sometimes referred to as a *compatible device context.*

2.3.1.4 Information

The information device context is used to retrieve default device data. For example, an application creates an information device context for a particular model of printer and then retrieves the default attributes. Because the system can retrieve device information without creating the structures, normally associated with the other types of device contexts, an information device context involves far less overhead and is created significantly faster than any of the other types.

2.4 Object

Objects are defined mathematically as a series of points joined by lines. The graphical elements in a vector file are called objects. Each object is a self-contained entity, with properties such as color, shape, outline, size, and position on the screen included in its definition.

Since each object is a self-contained entity, so its properties can be changed over and over again while maintaining its original clarity and crispness without affecting other objects in the drawing. These characteristics are ideal for text designing, in which the design process often requires individual objects to be created and manipulated.

2.4.1 Filling and Outlining Object

The object's outline is the line that surrounds the object. The fill is the color or pattern contained in the object. When an object is added to the drawing, its attribute can be changed to Solid Color, Gradient, Texture and pattern.

2.4.1.1 Solid Color

Since solid colors are even-colored they allow to uniform fill. These colors communicated by scanners, monitors, and printers in order to achieve a consistent and accurate reproduction of the colors as desired. A basic understanding of the color spaces and color management of the equipment helps to achieve the precise color required for the project.

We all see color differently. Color is subjective to the human eye. Each device that interacts with project's file: the scanner, monitor, and printer may have a different color space. For example, a color that is visible to the human eye may not be reproducible by the printer. Because there are so many color variations, a precise method for defining each color is required. For example once we find the perfect shade of light orange, we need to be able to reproduce that color and possibly tell others how to do the same. A color model defines that perfect shade of light orange by breaking it down into precise components that allow to accurately transmitting the information to other people and to the electronic devices used to create projects. A color model is a system used to organize and define colors according to a set of basic properties which are reproducible.

There are many different color models that define colors, for example, HSB, RGB, CMYK, and CIE Lab color models. The RGB and CMYK color models are only two of a

number of models developed to suit a variety of digital design and desktop publishing applications.

2.4.1.2 Gradient

A gradient fill or a ramp fill is a progression of colors that causes two or more colors to blend from one color to the others smoothly for adding depth and color in the drawing. When a shape is filled with a linear gradient, the color gradually changes as it moves across the shape. For example, suppose a horizontal gradient brush is created by specifying blue at the left edge of a shape and green at the right edge. When the shape is filled with the horizontal gradient brush, it will gradually change from blue to green as it moves from its left edge to the right edge. Similarly, a shape filled with a vertical gradient brush will change color as moved from its top to bottom.

When a shape is filled with a path gradient, then there are a variety of options for specifying how the colors change as it moves from one portion of the shape to another. One option is to have a center color and a boundary color so that the pixels change gradually from one color to the other as it moves from the middle of the shape towards the outer edges.

2.4.1.3 Pattern

A pattern is a simple picture composed of only "on" and "off" pixels. The two colors included in the bitmap are black and white. A bitmap pattern is a regular color picture. These bitmaps vary in complexity, and it is best to use less complex bitmaps for fill patterns, because complex ones are memory-intensive and slow to display.

2.4.1.4 Texture

A texture is a random, fractally-generated color that is used to give a natural appearance for wood, clouds, stone, ripples, waves, and wrinkles, or create artificial patterns such as checkers, dots, lines, and swirls. Texture fills increase the size of a file and the time it takes to print.

2.4.2 Drawing Objects

Drawing Objects are the basic shapes used for drawing i.e. Line, Circle, Ellipse and Rectangle.

2.4.3 Moving Objects

Moving Object is to change the position of an object on the screen. The easiest way to move and position the object is to drag and drop.

Chapter 3

System Analysis

3. System Analysis

Analysis is the foremost part of project development. Most of the time spent in project development is dedicated to analysis. System analysis was done using prototyping and object oriented methodology.

3.1 Analysis

Analysis plays a significant role in making of software. There are two main parts of the analysis.

- Requirement Analysis
- Domain Analysis

3.1.1 Requirement Analysis

The rationale of analysis is to provide a model of the system's behavior. In conducting the project, Object Oriented approach is adopted. Object oriented analysis is a method of analysis that examines the requirements from the perspective of the classes and objects found in the vocabulary of the problem domain. In requirement analysis we define use cases diagram containing use cases, actors. A first step in analysis is to extract scenarios, or *use cases* that describe the behavior of a system from an external user's perspective.

3.1.2 Domain Analysis

Conceptual domain analysis yields common ground for each specific analysis. Object Oriented analysis notions lend themselves for capturing generic concepts at multiple levels of granularity. Ensembles, sub ensembles, classes, and generic relationships are all candidates for describing an application domain. A requirements domain analysis may lead to an OO domain engineering effort. This entails the construction of design fragments of the generic

elements identified by a requirements domain analysis. These designs can be implemented and added to a domain-specific code library.

3.2 Steps for Object Oriented Analysis:

- Obtain ``complete" requirements.
- Describe system-context interaction.
- Delineate subsystems.
- Develop vocabulary by identifying instances with their classes, ensembles, and relationships.
- Elaborate classes and relationships by defining their generic static structure and describing their generic dynamic dimension.
- Construct a model in which the dynamics of objects are wired together.

These steps are connected by transformation -- elaboration relationships. The output of the last step, the model, feeds naturally into the design phase.

3.3 Use Cases

A use case is a specific way of using the system by using some part of the functionality. Each use case constitutes a complete course of events initiated by an actor and it specifies the interaction that takes place between an actor and the system. A use case is thus a special sequence of related transactions performed by an actor and the system in a dialogue. The collected use cases specify all the existing ways of using the system.

3.3.1 Use Case Analysis

Use case analysis is performed to identify portion of system performing specific task. In use case analysis use cases, actors interacting with those use cases, and boundaries are identified. A use case comprises a course of events begun by an actor, and it specifies the interaction between actor and the system. All the use cases specify existing ways to use the whole system. It is interaction of actors with external or other system with system being

designed in order to achieve a goal. Use case describes the functionality of the product to be constructed.

3.3.2 Actors

There is one actor in this use case diagram

- User

3.3.3 Use Case Expanded Format

There are fifteen use cases in this domain each of which shows its functionality. These are as follows.

- Rectangle
- Circle
- Ellipse
- Line
- Text
- Layered Text
- Undo
- Redo
- Cut
- Copy
- Paste
- Delete
- Duplicate
- Order
- Break Text

The use case diagram of our project is shown in Figure 3.1 followed by the detailed information in expanded format of each use case.

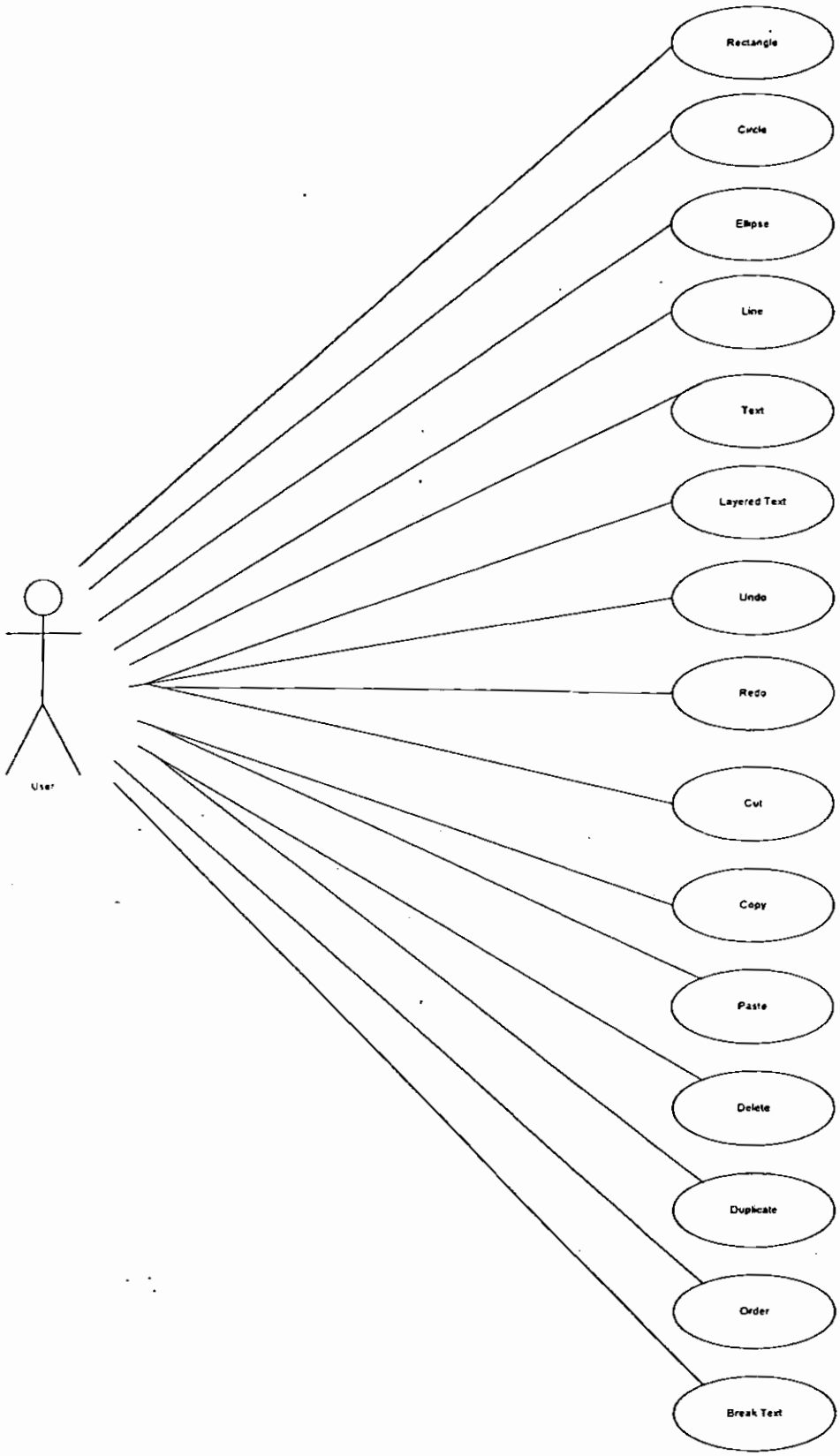


Figure 3.1 Use Case diagram of UrduKashishStyler

3.3.3.1 Use Case Rectangle

Actors: User

Purpose: To draw the rectangle.

Overview: User draws the rectangle on the drawing area.

Type: Real and Primary

Preconditions

- Rectangle Tool is selected from Drawing toolbar.

Post conditions

- The rectangle is drawn on the drawing area.

Initiation

This use case is initiated when user moves the mouse while keeping the left mouse button down.

Navigation

Action	System Response
1. User presses the left mouse button.	2. Set the point as starting of rectangle.
3. The user moves the mouse while keeping the left mouse button down.	4. A temporary rectangle is drawn on the screen from where the user pressed left mouse button to the current

	mouse position.
5. The user releases the left mouse button.	6. Rectangle with the selected attributes is drawn.

Alternative courses

5.a User presses the right mouse button, rectangle draw action is canceled and temporary rectangle is removed from the screen.

3.3.3.2 Use case Circle

Actors: User

Purpose: To draw the circle.

Overview: User draws the circle on the drawing area.

Type: Real and Primary

Preconditions

- Circle Tool is selected from drawing toolbar.

Post conditions

- The circle is drawn on the drawing area.

Initiation

This use case is initiated when user moves the mouse while keeping the left mouse button down.

Navigation

Action	System Response
1. User presses the left mouse button.	2. Set the point as center of circle.
3. The user moves the mouse while keeping the left mouse button down.	4. A temporary circle is drawn on the screen with dotted line, using the radius point from where the user pressed left mouse button to the current mouse position.
5. The user releases the left mouse button.	6. Circle with the selected attributes is drawn.

Alternative courses

5.a User presses the right mouse button, circle draw action is canceled and temporary circle is removed from the screen.

3.3.3.3 Use case Ellipse

Actors: User

Purpose: To draw the Ellipse.

Overview: User draws the ellipse on the drawing area.

Type: Real and Primary

Pre conditions

- Ellipse Tool is selected from drawing toolbar.

Post conditions

- The ellipse is drawn on the drawing area.

Initiation

This use case is initiated when user moves the mouse while keeping the left mouse button down.

Navigation

Action	System Response
1. User presses the left mouse button.	2. Set the point as starting of ellipse.
3. The user moves the mouse while keeping the left mouse button down.	4. A temporary ellipse is drawn on the screen, with dotted line, from where the user pressed left mouse button to the current mouse position.
5. The user releases the left mouse button.	6. Ellipse with the selected attributes is drawn.

T-845

Alternative courses

5.a User presses the right mouse button, line draw action is canceled and temporary line is removed from the screen.

3.3.3.4 Use case Line

Actors: User

Purpose: To draw the line.

Overview: User draws the line on the drawing area.

Type: Real and Primary

Preconditions

- Line Tool is selected from Drawing toolbar.

Post conditions

- The line is drawn on the drawing area.

Initiation

This use case is initiated when user moves the mouse while keeping the left mouse button down.

Navigation

Action	System Response
1. User presses the left mouse button.	2. Set the point as starting of line.

3. The user moves the mouse while keeping the left mouse button down.	4. A temporary straight line is drawn on the screen from where the user pressed left mouse button to the current mouse position.
5. The user releases the left mouse button.	6. Line with the selected attributes is drawn.

Alternative courses

5.a User presses the right mouse button, line draw action is canceled and temporary line is removed from the screen.

3.3.3.5 Use case Text

Actors: User

Purpose: To write the text.

Overview: User can write the text for designing.

Type: Real and Primary

Precondition

- Text tool is selected from Drawing toolbar.

Post condition

- Urdu text is available in UrduKashishStyler for designing.

Initiation

This use case is initiated when user clicks on the drawing area of the screen.

Navigation

Action	System Response
1. User clicks on the drawing area of the screen.	2. Kashish Urdu Editor appears for writing text.
3. User enters the text in the Urdu Kashish Editor.	4. Text will be displayed in the text area of dialog box.
5. Click on OK button.	6. The written text will be available in the Urdu Kashish Styler for designing.

Alternate Courses

1.a If the text was selected before click on Text button, then by default the selected text will be available in the Kashish Urdu Editor dialog box for modification.

5.a If user clicks on cancel button, the entered text will not be available in the UrduKashishStyler for designing.

3.3.3.6 Use case Layered Text

Actors: User

Purpose: To write Layered text.

Type: Real and Primary

Precondition

- Desired Layer Text Tool is selected from Drawing toolbar.

Post condition

- The text is written in selected layer.

Initiation

This use case is initiated when user clicks on the drawing area of screen.

Navigation

When user clicks on the drawing area of screen, a Text Art Gallery appears with different samples. If the text was selected before the click on screen, then the pattern of selected sample will be applied on the selected text, otherwise an Urdu Edit box will appear, where the user will write the text and the pattern of selected sample will be applied on this text. The user can also change the attributes i.e. line color, line style, line width and Fill of the selected sample.

3.3.3.7 Use case Undo

Actors: User

Purpose: To reverse the last action performed.

Overview: Gives user, the freedom and flexibility to experiment and be creative without worrying about permanently altering the drawings or having to start over. If you make a change to your document, then wish you hadn't, you can reverse the change.

Type: Real and Primary

Preconditions

- At least one altering in done on the drawing area.

Post conditions

- Reverses the last change.

Initiation

This use case is initiated when user gives the undo command.

Navigation

Action	System Response
1. User presses the Undo button.	2. Reverses the last action performed. 3. Updates the user drawing area.

Alternative courses

1. a No altering or change is done in the document, so not any of the action is performed.

3.3.3.8 Use case Redo

Actors: User

Purpose: Restores changes reversed by the Undo command.

Overview: Gives user, the facility to restore changes reversed by the Undo command. It is available immediately after you click the Undo command.

Type: Real and Primary

Preconditions

- At least one Undo command is performed by the user.

Post conditions

- Restores the changes.

Initiation

This use case is initiated when user gives the Redo command.

Navigation

Action	System Response
1. User presses the Redo button.	2. Restores the last action performed. 3. Updates the user drawing area.

Alternative courses

1. a No Undo command is performed by the user, so not any of the action is performed.

3.3.3.9 Use case Cut

Actors: User

Purpose: Removes selected object or text from drawing area and places them on the clipboard.

Type: Real and Primary

Preconditions

- At least one of the object is selected.

Post conditions

- The selected object is removed from the drawing area and placed on the clipboard.

Initiation

This use case is initiated when user gives the Cut command.

Navigation

Action	System Response
1. User presses the Cut button.	2. Place the selected object on the clipboard. 3. Updates the user drawing area.

3.3.3.10 Use case Copy

Actors: User

Purpose: Copies selected object or text to the clipboard.

Type: Real and Primary

Preconditions

- At least one of the objects is selected.

Post conditions

- The selected object is placed on the clipboard to paste anywhere on the drawing area.

Initiation

This use case is initiated when user gives the Copy command.

Navigation

Action	System Response
1. User presses the Copy button.	2. Place the selected object on the clipboard.

3.3.3.11 Use case Paste

Actors: User

Purpose: Copies the text or objects to the drawing area that have been copied or cut to the clipboard.

Type: Real and Primary

Preconditions

- At least one of the objects is placed on the clipboard.

Post conditions

- The object or text placed on the clipboard will be copied on the drawing area.

Initiation

This use case is initiated when user gives the Paste command.

Navigation

Action	System Response
1. User presses the Paste button.	2. Paste the text or object placed on the clipboard, to the drawing area.

3.3.3.12 Use case Delete

Actors: User

Purpose: To remove selected object or text without placing a copy on the clipboard.

Type: Real and Primary

Preconditions

- At least one object or text is selected from drawing area.

Post conditions

- The selected object is deleted.

Initiation

This use case is initiated when user gives the delete command.

Navigation

Action	System Response
1. User presses the Delete button.	2. Deletes the selected object from the list of objects. 3. Updates the user drawing area.

3.3.3.13 Use case Duplicate

Actors: User

Purpose: To create a copy of the selected object.

Type: Real and Primary

Preconditions

- At least one object or text is selected from drawing area.

Post conditions

- A copy of the selected object is created.

Initiation

This use case is initiated when user gives the duplicate command.

Navigation

Action	System Response
1. User presses the Duplicate button.	2. A new object of selected type will be added in the list of objects. 3. Updates the user drawing area.

3.3.3.14 Use case Order

Actors: User

Purpose: To change the sequence number of object.

Overview: Order is used to change the sequence of objects created in the Image Window. This order determines the relationship between objects and, therefore, the appearance of your image. The first object you create appears on the bottom and the last object appears on the top. You can use the Order commands to place the objects where you want them.

Type: Real and Primary

Preconditions

- At least one object or text is selected from drawing area.

Post conditions

- Selected object is drawn according to the selected order.

Initiation

This use case is initiated when user gives the Order command.

Navigation

Action	System Response
1. User presses the Order (to Front, to Back, one step Forward, one step Back, inFront of...) button.	2. The sequence of selected object is changed. 3. Updates the user drawing area.

3.3.3.15 Use case Break Text

Actors: User

Purpose: To break the text.

Overview: User can break the text so individual words can be placed anywhere on the screen.

Type: Real and Primary

Preconditions

- Break Text Tool is selected from Drawing toolbar.

Post conditions

- The individual words of text can be placed anywhere on the drawing area.

Initiation

This use case is initiated when user clicks on the text.

Navigation

Action	System Response
1. User presses the left mouse button on the text.	2. Text is broken.
3. The user moves the word of text anywhere on the screen.	4. Drawing screen area is updated.

Alternative courses

1.a Click is not on the text. Try again.

3.a User doesn't move the text and let it remain on the same location.

3.4 Domain Analysis

In domain analysis we represents concepts of our project .Discuses the functionality of the project by representing conceptual diagram .Which contains main concepts and their relations and their attributes.

3.4.1 Conceptual Diagram

Conceptual Model is a quintessential step in analysis or investigation, it is a decomposition of the problem into individual Objects (called concepts), and the things we are aware of. In addition to that creating a Conceptual Model also aids in clarifying the terminology or vocabulary of the domain. A Conceptual Model is a description of things in a real world problem domain, that is, it is not a Model of software design.

Figure 3.2 show the conceptual model and explains the main concept of the UrduKashishSyler software. The Figure shows its working, relation between different functions, relation between Concepts, their dependences and their corporations.

We have thirteen concepts in UrduKashishStyler problem domain which explain the main operations and idea of the project.

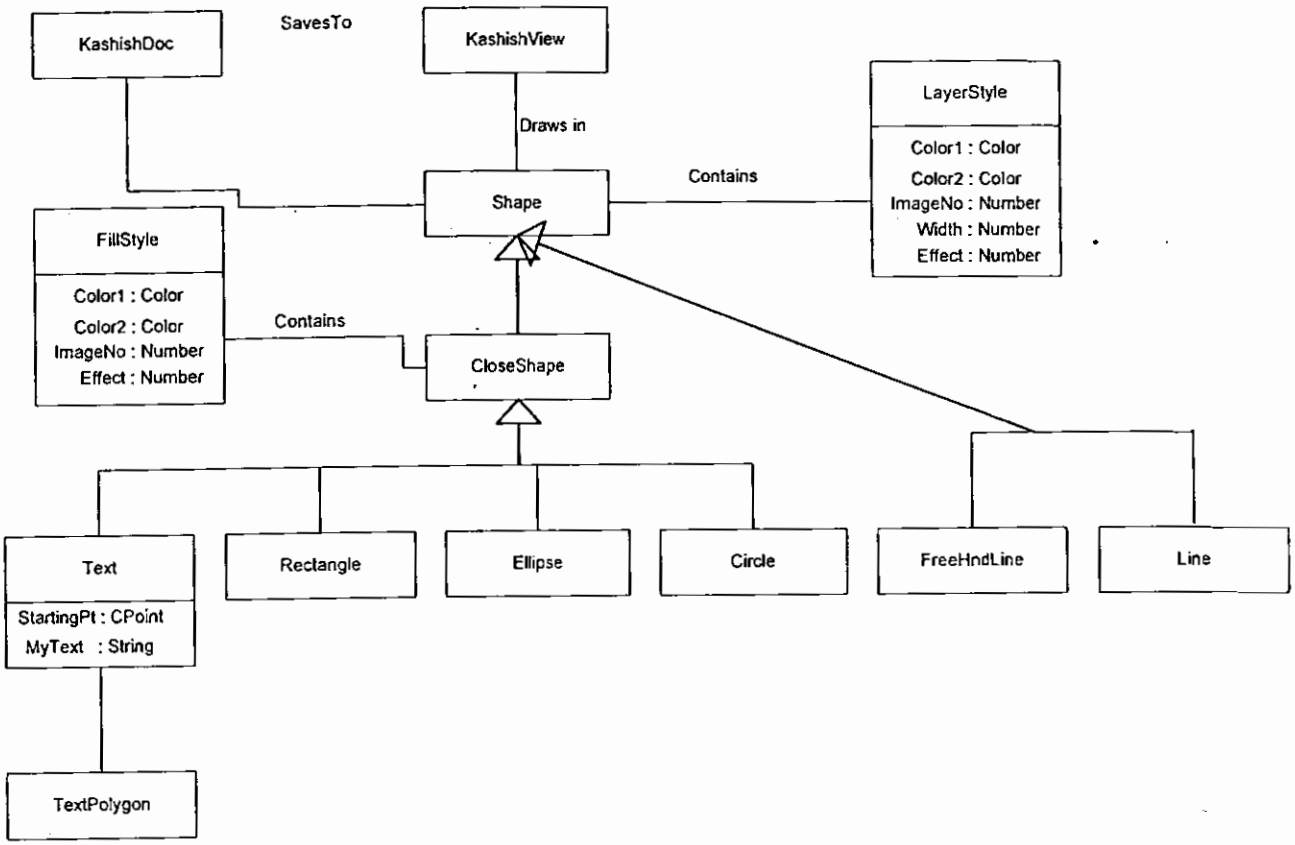


Figure 3.2 Conceptual Model of UrduKashishStyler

3.4.2 Concepts

There are thirteen concept in UrduKashishStyler which explains the main working and concept of the project.

- KashishDoc
- KashishView
- FillStyle
- LayerStyle
- Shape

- CloseShape
- Text
- Rectangle
- Ellipse
- FreeHandLine
- Circle
- Line
- TextPolygon

Chapter 4

System Design

4. System Design

The purpose of design is to create architecture for the evolving implementations. Object oriented design is a method of design encompassing the process of objects oriented decomposition and a notation for depicting logical and physical as well as static and dynamic models of system under design.

The design phase focuses on defining the software to implement the application. The design object is to produce a model of the system, which can be used later to build the system. The design goal is to find the best possible design within the limitations imposed by the requirement and the physical and social environment in which the system will operate.

4.1 Activity Diagrams

It gives the pictorial representation of algorithm for the function. Activity diagram is used to represent activities present in use cases. Basic need is that we want to make procedural design in Unified Modeling Language (UML). Operations in use cases in sequence are represented in activity diagram. Activity diagram are useful when we want to describe a behavior which is parallel, or when we want to show how behaviors in several use cases interact. The activity diagrams are described as follows.

- The process of drawing a shape by the user is given in figure 4.1
- The process of writing text by the user is shown in figure 4.2
- The process of printing is shown in figure 4.3

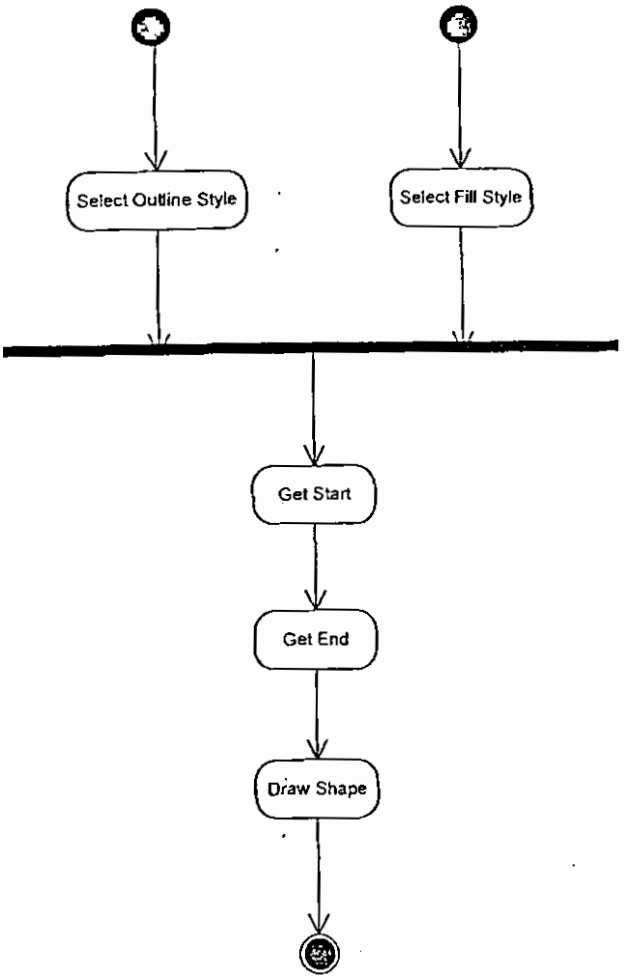


Figure 4.1 Activity Diagram of Drawing a Shape

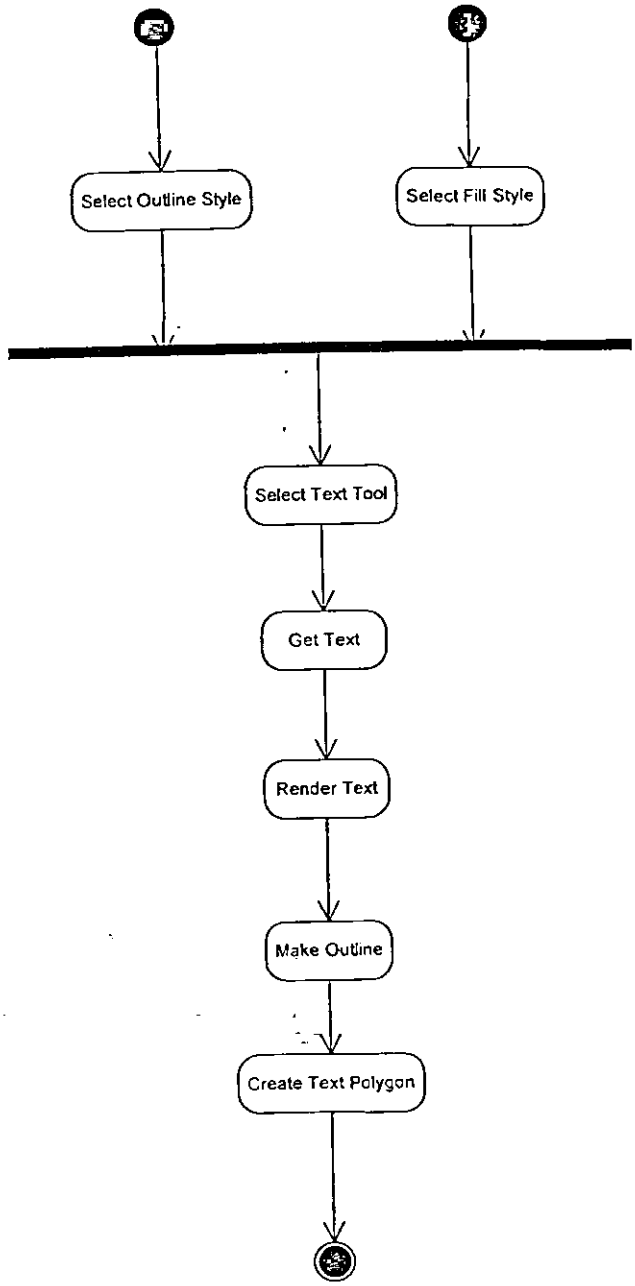


Figure 4.2 Activity Diagram of Writing Text

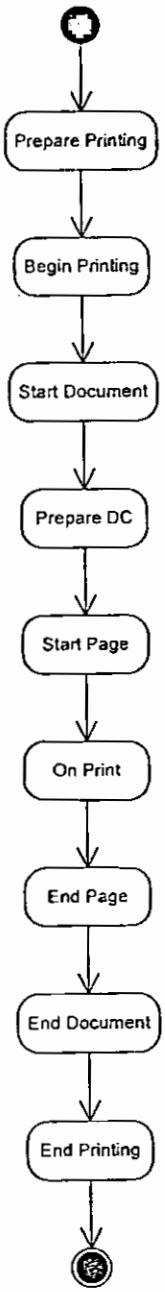


Figure 4.3 Activity Diagram of Printing Process

4.2 Class

A class implements one or more interfaces. Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations. Class stands for a family of objects that have something in common. A class is not to be equated with a set of objects, although at any moment we can consider the set of instances that belong to the class. A class may be seen as what all these sets have in common. In technical terminology, a class stands for the intension of a particular characterization of entities, while the set of objects that conform to such a characterization in a certain period is known as the extension.

The development phase produces candidate classes and relationships. After selecting concise and evocative names we must describe each class with attributes. Although each class must have a unique name, classes should be distinguishable on the basis of their attribute characterizations. A rule of thumb is if two classes have identical attributes, then they are most likely the same. Class diagram of our project is shown in Figure 4.4.

4.2.1 Attribute

An attribute expresses an essential definitional feature that is shared by all instances of a class. A minimal characterization of an attribute consists of the value domain of the attribute and a name that explains the role or relationship that an attribute value has with respect to the instance to which it belongs. Multi valued attributes may be annotated with multiplicity characterizations. Defaults for an attribute value and/or multiplicity description can be formulated in this phase as well. Constraints can restrict attribute value combinations and/or refer to multiplicity descriptions.

Real-life entities are often described with words that indicate stable features. Most physical objects have features such as shape, weight, color, and type of material. Sometimes it is useful to indicate a default initial value for an attribute.

4.2.2 Relationships

Relationships help capture target system-specific knowledge by describing connections among different objects. Relationships may also be used to modify descriptions in the previous step. For example, when an attribute has a multiplicity range that includes zero, one may eliminate the attribute and represent this information as a relationship instead.

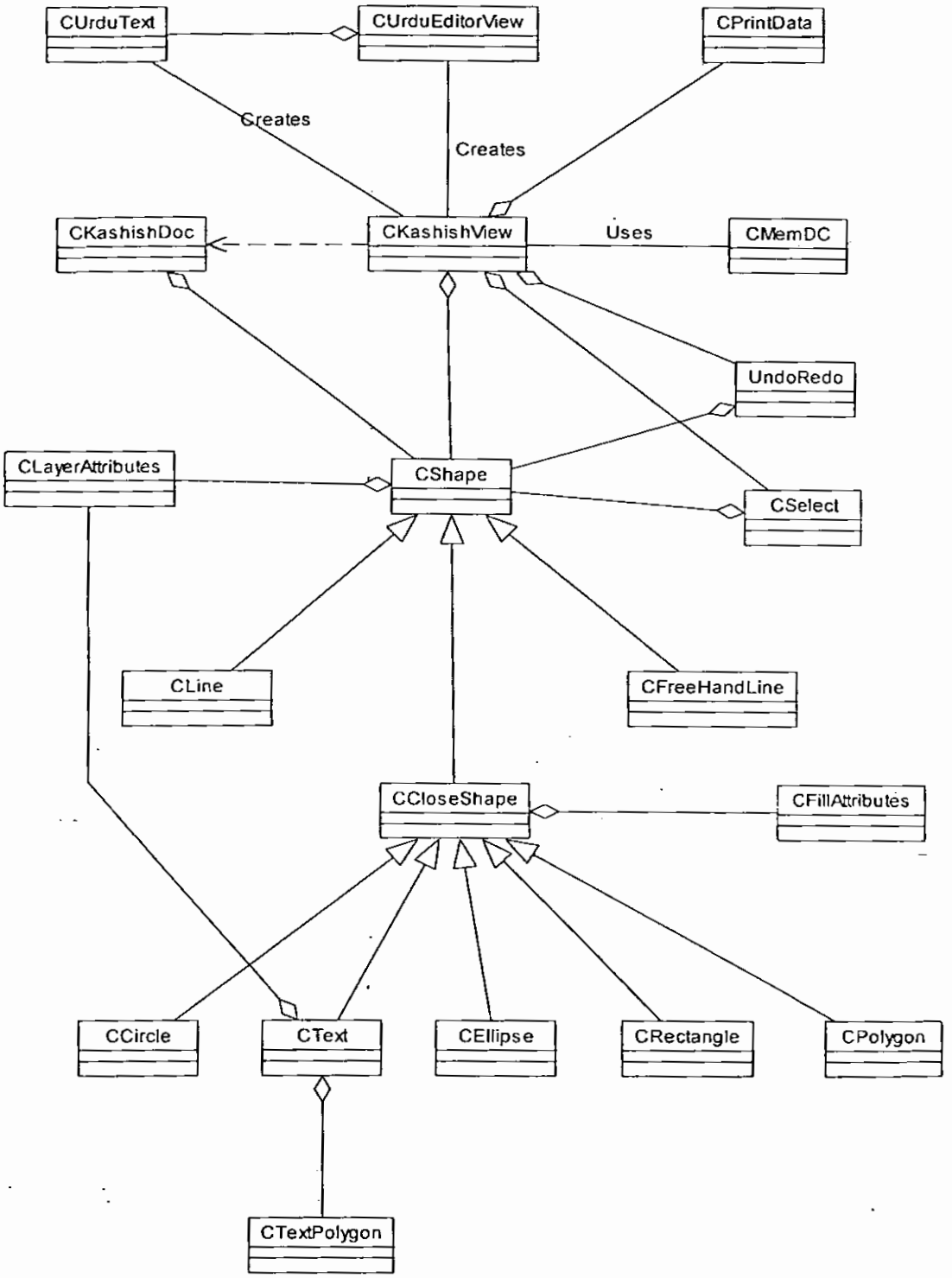


Figure 4.4 Class Diagram of UrduKashishStyler.

Now we will show a detailed view of each class.

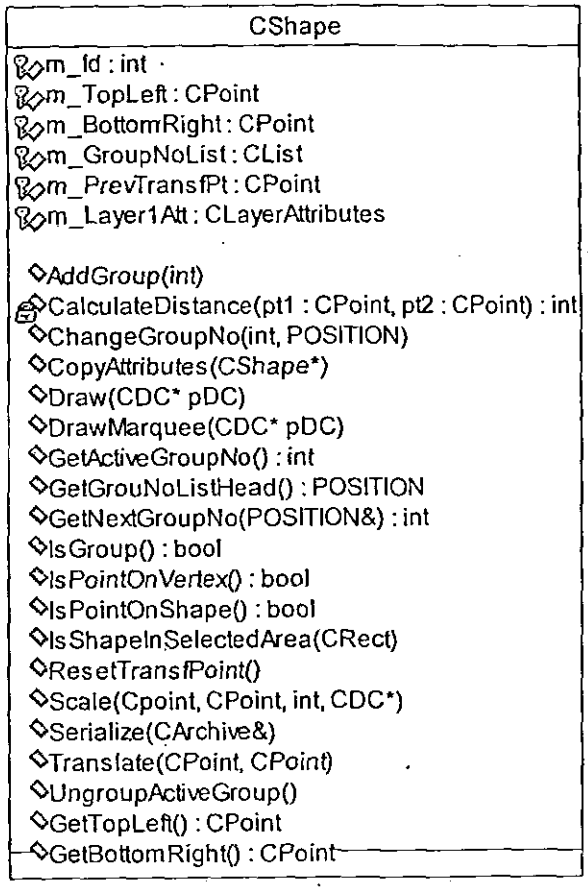


Figure 4.5 Class CShape.

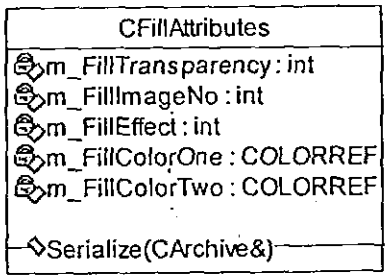
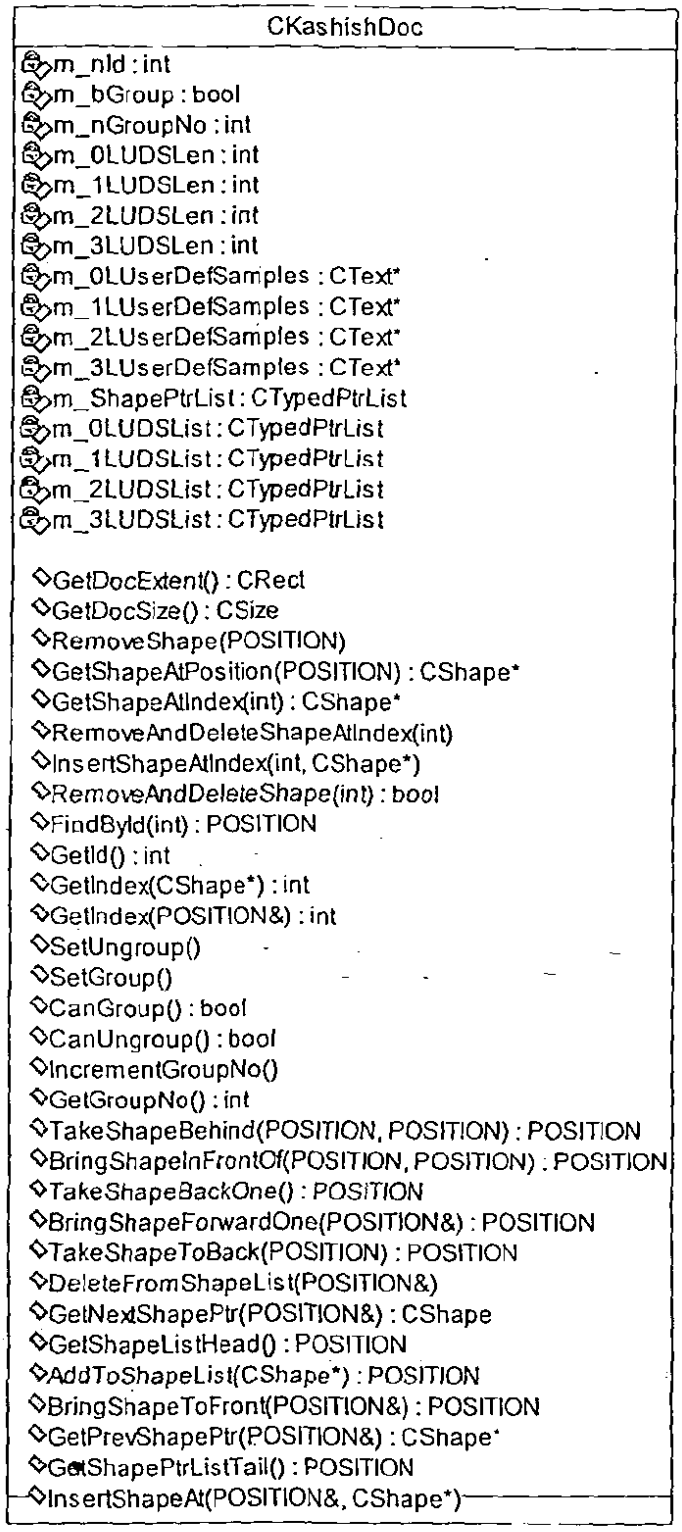


Figure 4.6 Class CFillAttributes.



. **Figure 4.7** Class CKashishDoc.

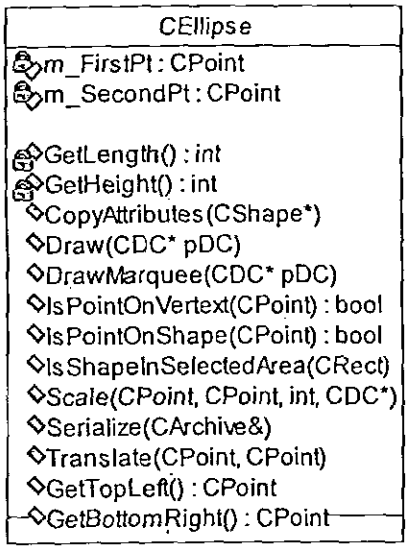


Figure 4.11 Class CEllipse.

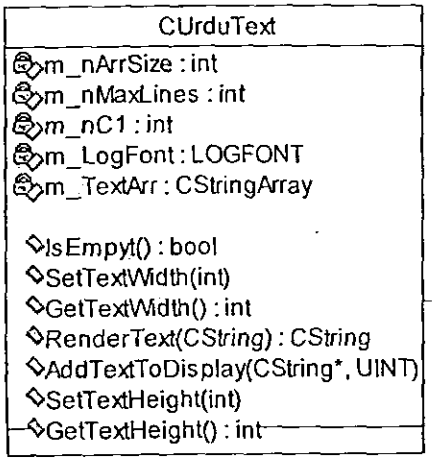


Figure 4.12 Class CUrduText

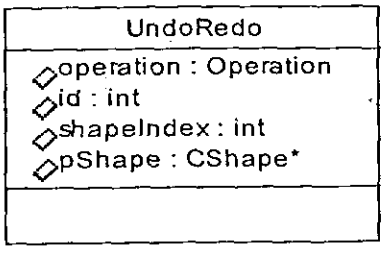


Figure 4.13 Struct UndoRedo

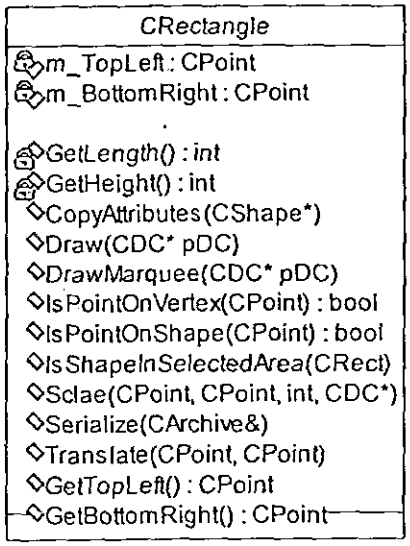


Figure 4.14 Class CRectangle.

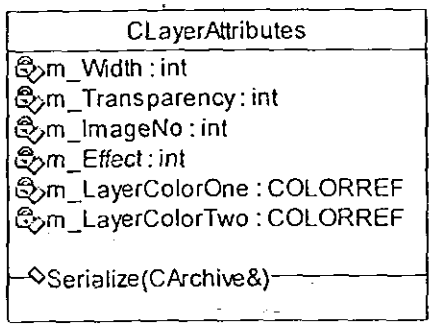


Figure 4.15 Class CLayerAttributes

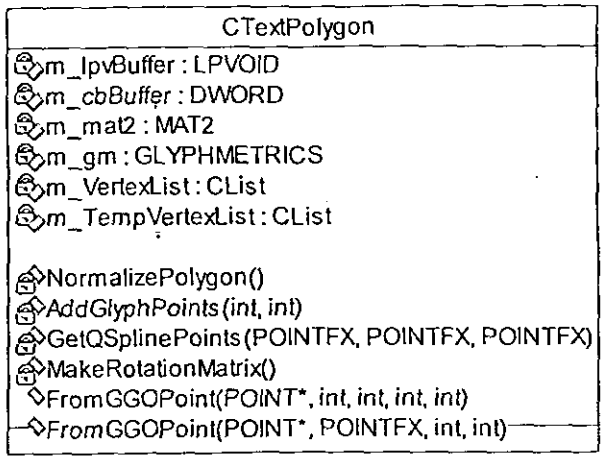


Figure 4.16 Class CTextPolygon.

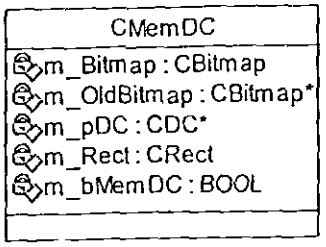


Figure 4.17 Class CMemDC.

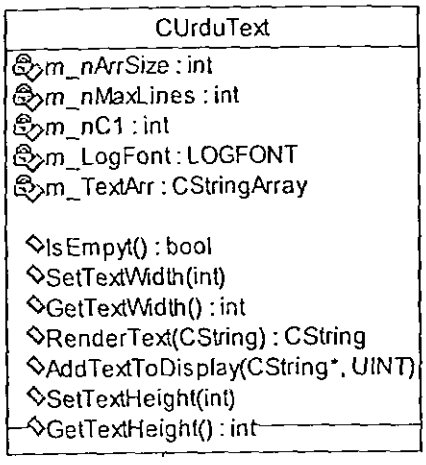


Figure 4.18 Class CUrduText.

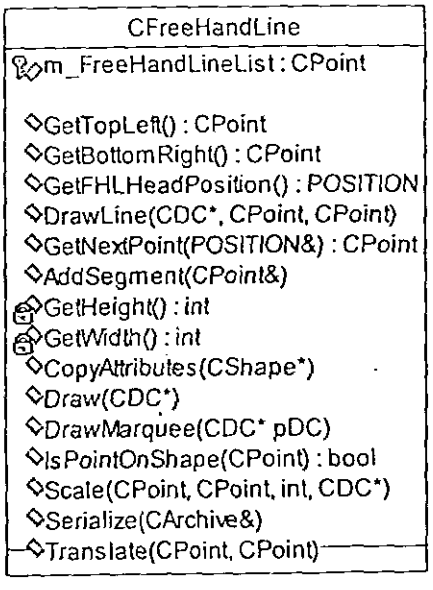


Figure 4.19 Class CFreeHandLine.

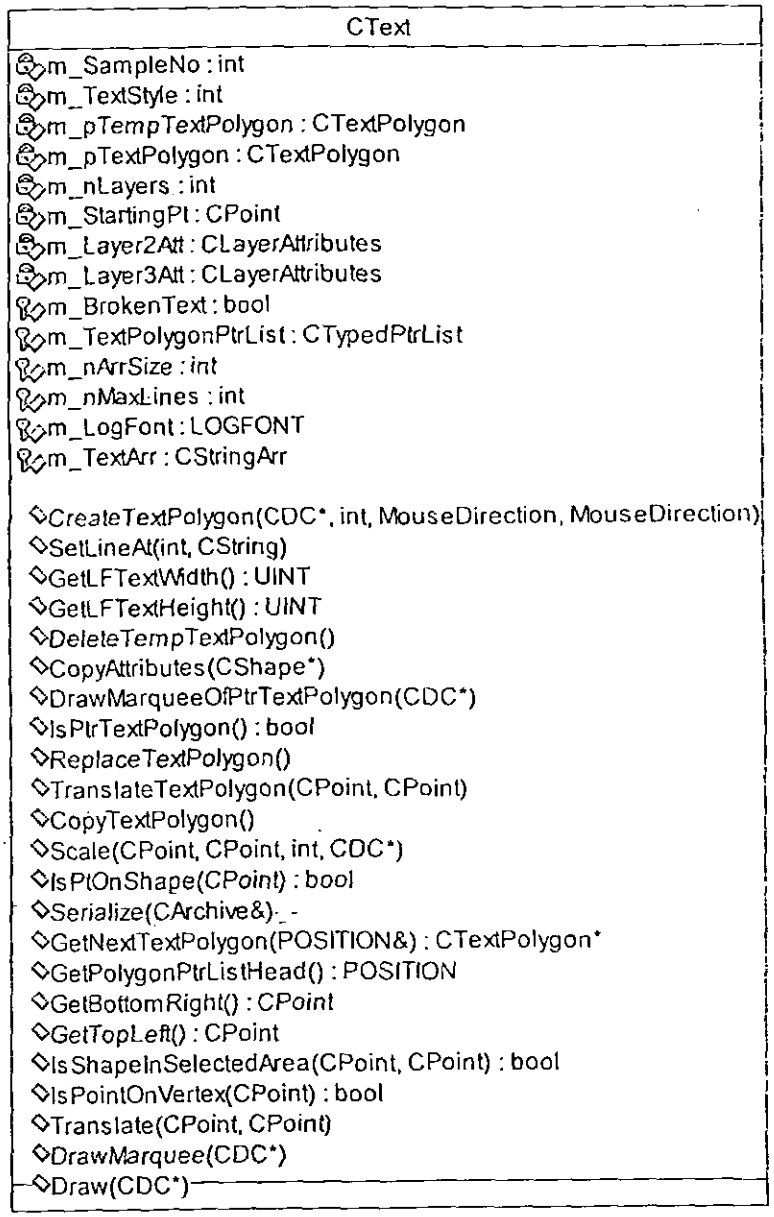
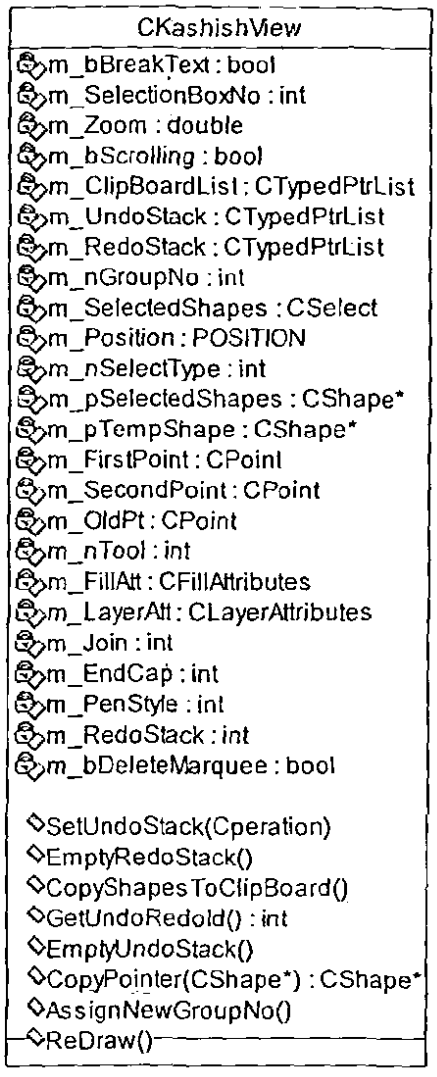


Figure 4.20 Class CText



. **Figure 4.21** Class CKashishView.

4.3 Sequence Diagram

Sequence diagrams are used to show the flow of functionality through a use case. For one use case diagram there can be multiple sequence diagrams. For alternate course of actions there are separate sequence diagrams. Sequence diagrams are time dependent and tell which operation will be executed first. Sequence diagrams define a pattern of interaction among objects arranged in chronological order. These diagrams show the objects participating in interaction by the order of their life times and the messages being sent from one object to the other. The following are sequence diagrams.

4.3.1 Drawing a Line

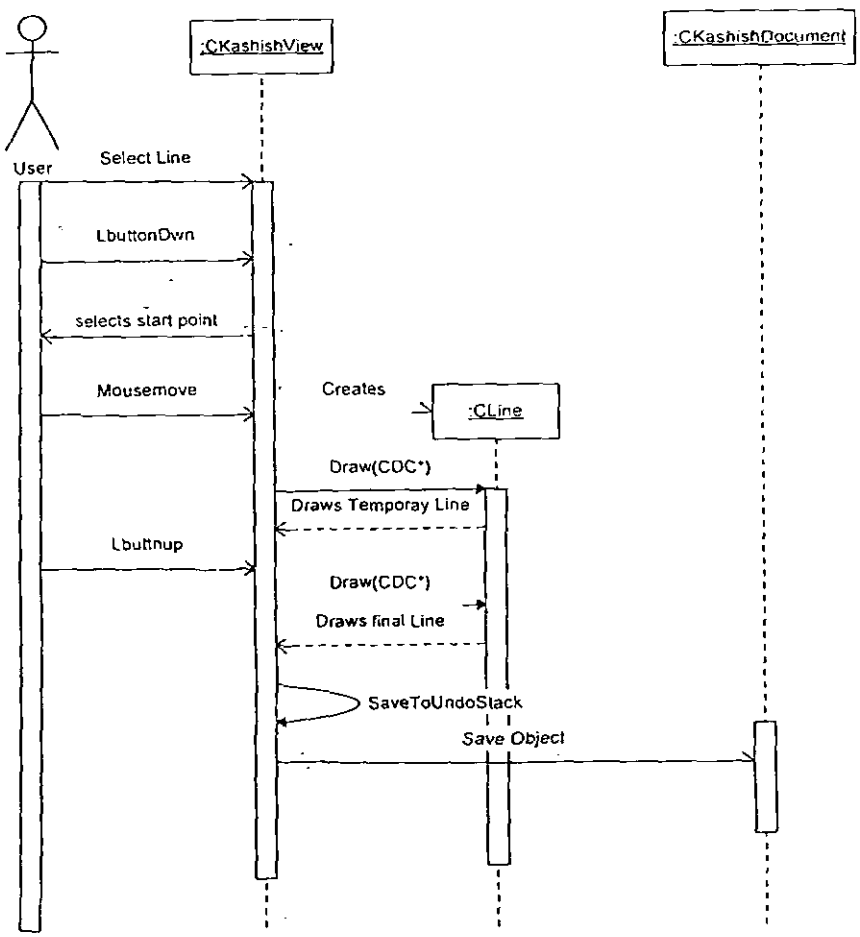


Figure 4.23 Sequence diagram of Drawing a line.

4.3.2 Drawing a Rectangle

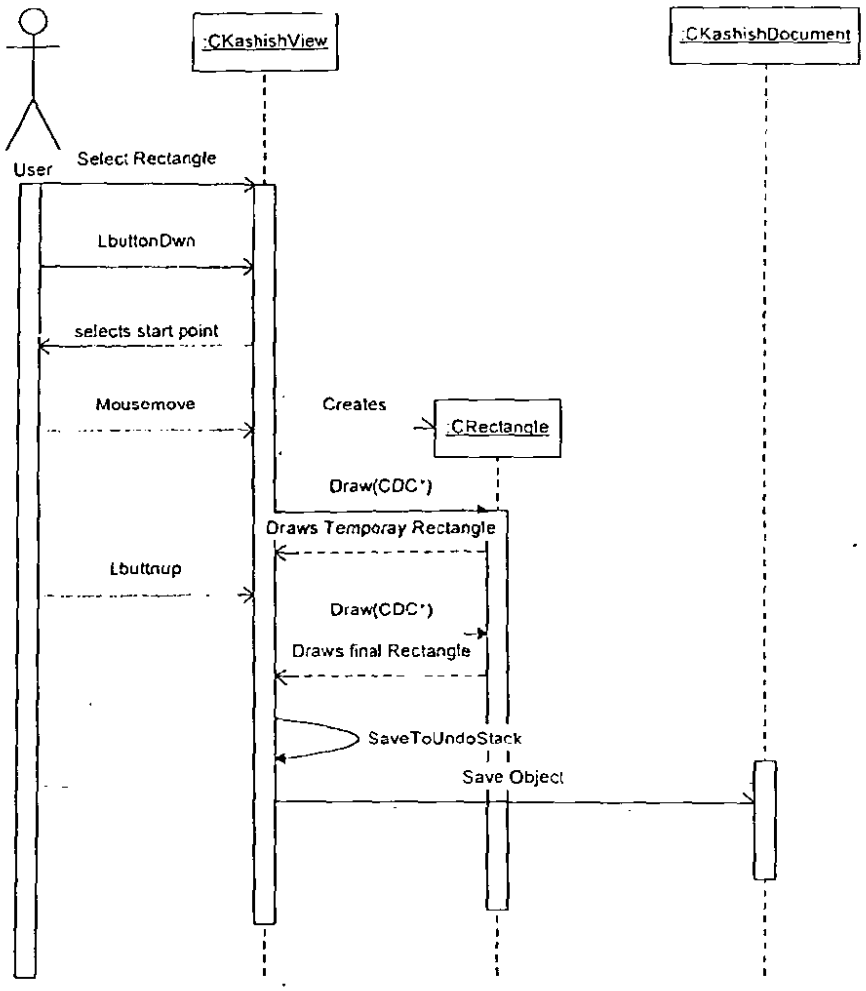


Figure 4.24 Sequence diagram of Drawing Rectangle.

4.3.3 Drawing an Ellipse

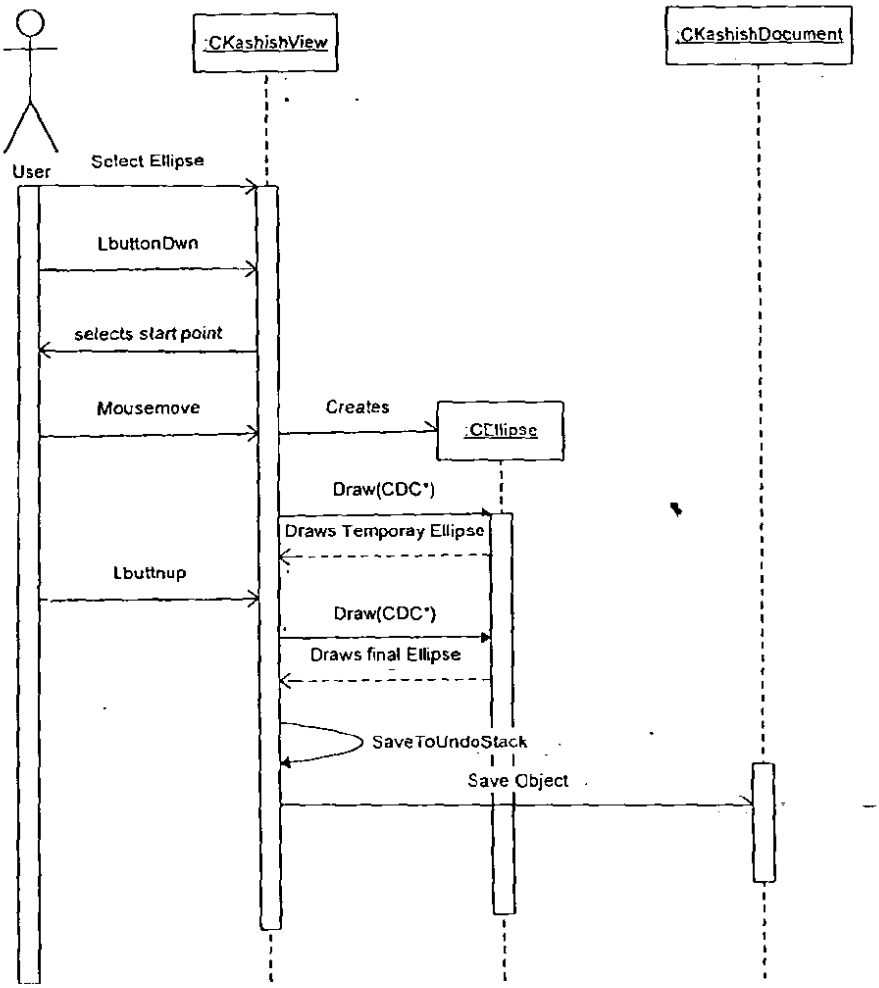


Figure 4.25 Sequence diagram of Drawing an Ellipse.

4.3.4 Drawing a Circle

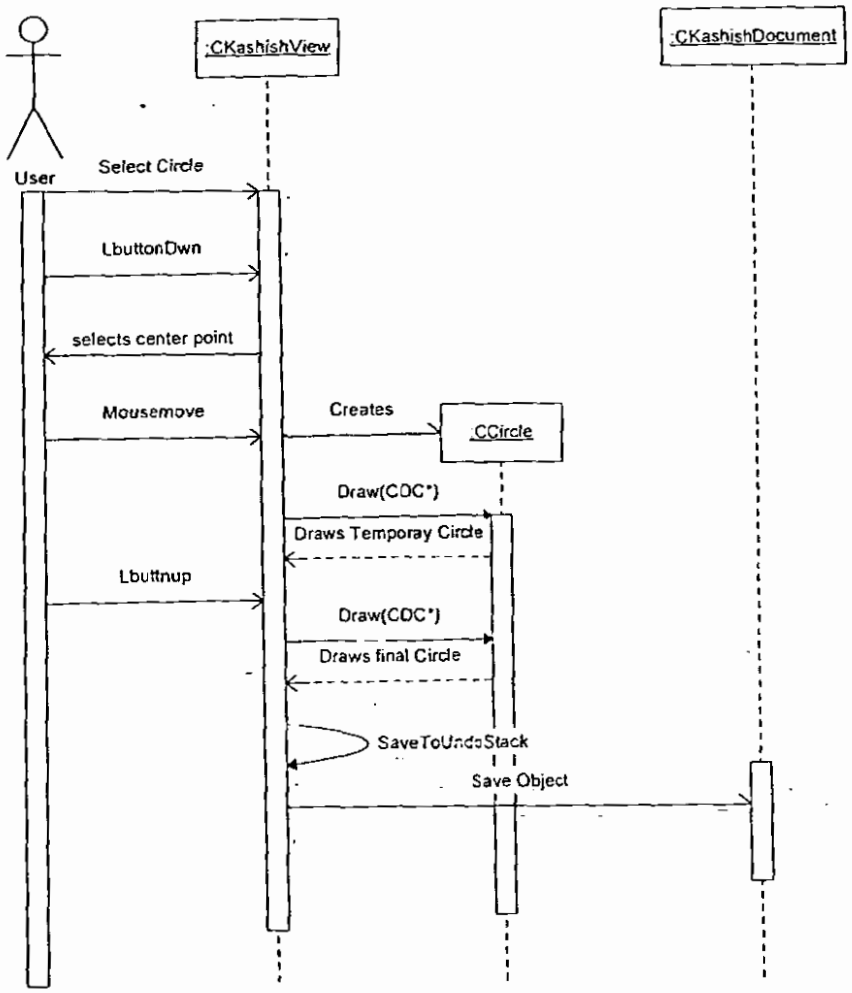


Figure 4.26 Sequence diagram of Drawing a Circle.

4.3.5 Selection Process

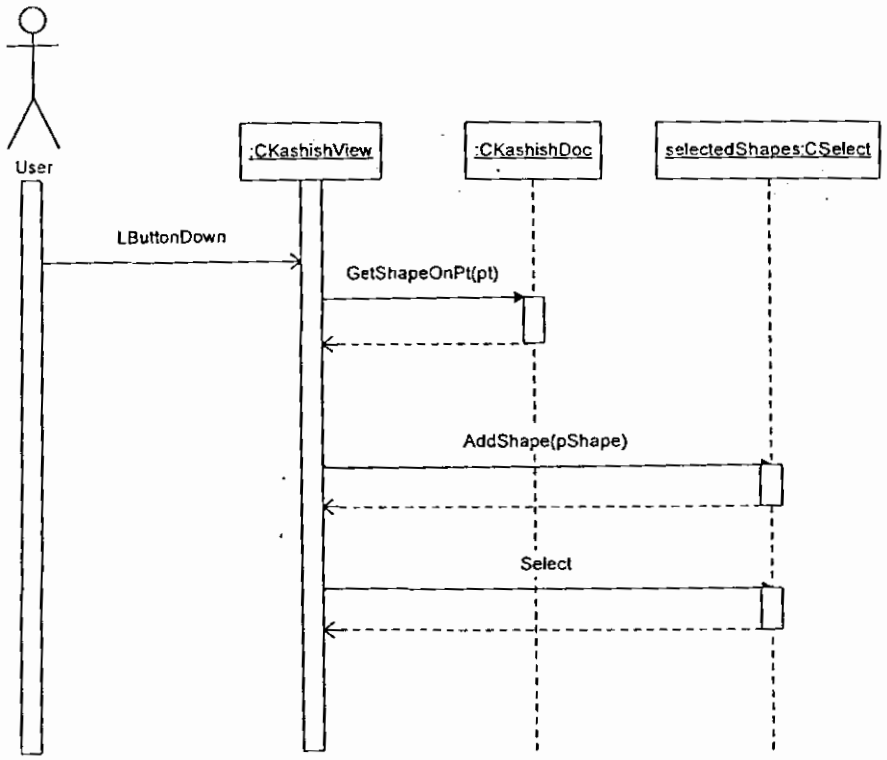


Figure 4.27 Sequence diagram of Selection Process.

4.3.6 Translate Process

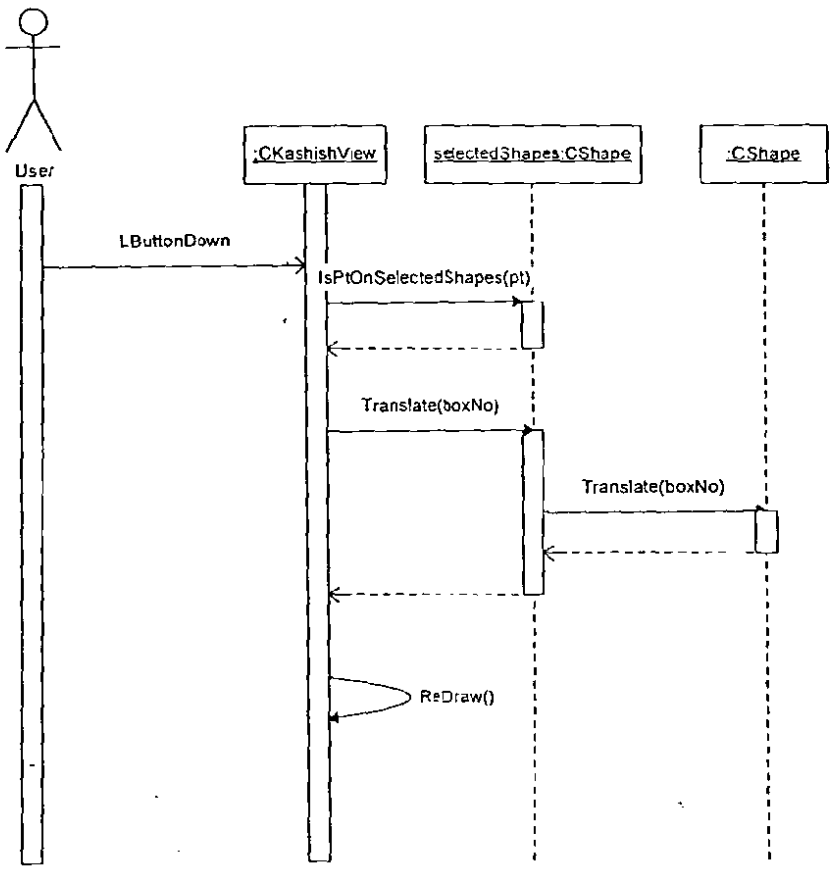


Figure 4.29 Sequence diagram of Translate Process.

4.3.7 Group Process

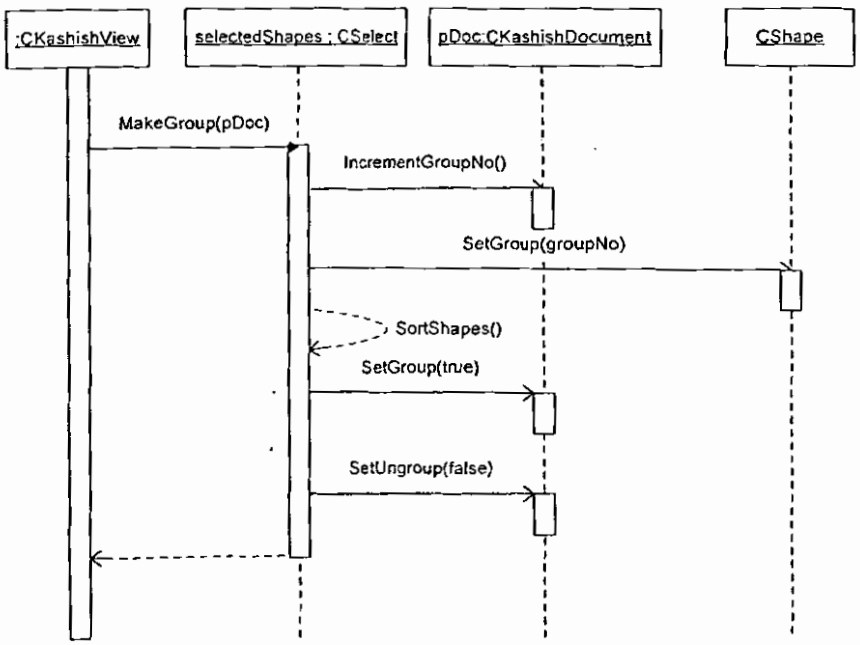


Figure 4.30 Sequence diagram of Group Process.

Chapter 5

Implementation

5. Implementation

UrduKashishStyler is a comprehensive vector-based drawing and graphic-design program for the composers. The software is composed of two major parts.

- Urdu Editor.
- Graphical Editor.

An editor is developed to write Urdu which is than used in composing and designing. So we will first explain the working of Urdu Editor.

5.1 Urdu Editor

Urdu editor is composed of three parts.

- Font file
- Word Processor
- Output Screen

These three parts and there interaction with each other is shown in figure 5.1.

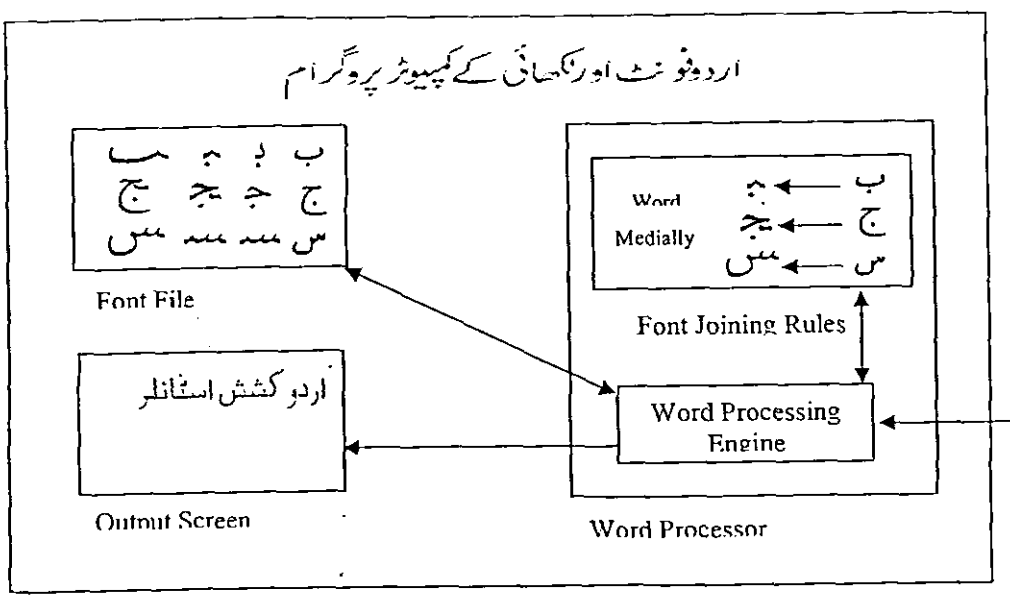


Figure 5.1 Parts of Urdu editor.

5.1.1 Font File

Urdu language has thirty-four characters. Each character has more than one form. All these forms of word are stored in a font file called True type font file. These forms of characters are called Glyph. A Glyph is the representation of a character. A detailed analysis of Urdu language shows that its characters have following forms shown in figure 5.2.

	Isolate	Initial	Middle	Final
ALIF	ا			ا
BAY	ب	بـ	ـبـ	ـب
PAY	پ	پـ	ـپـ	ـپ
TAY	ت	تـ	ـتـ	ـت
TTAY	ٹ	ٹـ	ـٹـ	ـٹ
SAY	ث	ثـ	ـثـ	ـث
JEEM	ج	جـ	ـجـ	ـج
CHAY	چ	چـ	ـچـ	ـچ
HAY	ح	حـ	ـحـ	ـح
KHAY	خ	خـ	ـخـ	ـخ
DAL	د			ـد
DDAL	ڈ			ـڈ
ZAAL	ز			ـز
RAY	ر			ـر

RRAY	ڑ			ڑ
SEEN	س	سہ	سہ	س
SHEEN	ش	شہ	شہ	ش
SUAT	ص	صہ	صہ	ص
ZUAT	ض	ضہ	ضہ	ض
TUAY	ط	طہ	طہ	ط
ZUAY	ظ	ظہ	ظہ	ظ
EAIN	ع	عہ	عہ	ع
GHAIN	غ	غہ	غہ	غ
FAY	ف	فہ	فہ	ف
KAAF	ك	کہ	کہ	ك
QAAF	ق	قہ	قہ	ق
GAAF	گ	گہ	گہ	گ
LAAM	ل	لہ	لہ	ل
MEEM	م	مہ	مہ	م
NOON	ن	نہ	نہ	ن
WOW	و			و
HAMZA	ہ	ہہ	ہہ	ہ

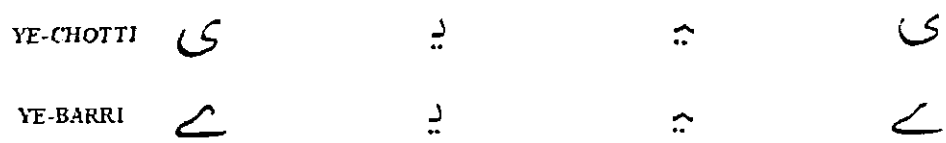


Figure 5.2 Table of Urdu chars and their possible forms.

In true type font file just information of shape of characters is present but no way is defined in the font file to join these characters. The developers of Urdu editor themselves write program in which the logic for joining these characters is defined often called as word processor on character rendering engine.

5.1.2 Word Processor

Word Processor is a program to join different characters. As discussed before, a character has different shapes. The decision of selection of shape of a character depends on

- Position of character in the word.
- Character before that character.
- Characters after that character.

All these joining rules are defined in the word processor.

The characters of Urdu can be adjusted at any of the four places in a word.

- Isolate
- Initial
- Middle
- Final

All the characters can not come at all the places. After analysis of Urdu language we have divided all the characters of Urdu into four separate groups. First group consist of those

characters which can not come at the beginning and in the middle of a word example of such characters are ALIF, DAL etc. These words have only two forms i.e. isolate and final. Second group comprises those characters which can come at any of the four places e.g. BAY, PAY etc. Third group consist of HAMZA which can not come at the end of a word.

Word processor gets a string of isolated character as input and selects the shape of the character depending on three rules described above. For example if the string passed to the word processor is Muhammad i.e. MEEM HAY MEEM DAAL. It will start with first MEEM check the character after it which is HAY, HAY has final form so selects initial form of MEEM then for HAY first check the character before it, which is MEEM and has initial form then check the character after HAY which is again MEEM and can have the final form so selects the middle form of HAY and this process continues until whole string is parsed.

5.1.3 Output Screen

The output of word processor is displayed by the output screen to the user.

5.2 Graphical Editor

The graphical editor consists of following tools.

- Select Tool.
- Rectangle Tool.
- Ellipse Tool.
- Circle Tool.
- Line Tool.
- Free Hand Tool.
- Text Tool.
- Zero Layer Text Tool.
- One Layer Text Tool.
- Two Layer Text Tool.
- Three Layer Text Tool.

- Break Text Tool.

5.2.1 Select Tool

Three types of actions can be carried out with the Select tool. Graphical objects can be selected, they can be moved, and they can be scaled.

5.2.1.1 Selection Actions

The select tool is used to select objects, so that they can be subjected to editing commands, such as *Cut*, copy or *Delete* etc.

When the Select tool is active, and user selects an object by moving the mouse so that the mouse cursor points to the object and then clicks the left mouse button. The object is then selected, and it takes on a selected appearance. 'Selection handles' appear around the object's bounding rectangle. These are small black rectangles that appear on the periphery of objects to show that they have been selected.

The second way to select single object is by using tab. When no object is selected and user presses tab, the object which is drawn first is selected. If the user presses tab again then the object which is drawn after that object gets selected and this process continues. First object is selected again after last object.

All types of graphic shapes display same patterns of selection handles. All UrduKashishStyler shapes display eight handles spaced around their bounding rectangles four at the corners and one in the middle of each side of the bounding rectangle.

Selection actions are not undoable. That is, the Edit menu's Undo command will apply to a prior action, not to the selection action that user just made. The reason for this is that current selections are not remembered when a document is saved. Since selection does not change the document, it does not need to be undoable.

Of course, it is very easy in practice to 'undo' a selection, simply by selecting something else, or by selecting nothing. If user clicks in some part of a scene in which there are no

graphics, then the selection will be set to nothing.

5.2.1.2 Multiple Selections

Multiple graphic objects can be selected using the Select tool. One way to select several objects is to drag the mouse through a rectangular area that completely encloses the objects that are to be selected. For example, user can put down the mouse button to the left and above the set of objects to be selected, and then drag the mouse to the right and below the entire set before releasing the mouse button. As the mouse is dragged, a dotted line selection rectangle is traced out. All the objects fully enclosed in this rectangle will be selected when the mouse button is released.

The second way to select multiple graphical objects is to hold down one of the keyboard shift keys while clicking the mouse on the second and subsequent objects to be selected. Shift-click has the effect of adding an unselected object to the set of currently selected objects. If one shift-click on an already selected object, the action will have the effect of *deselecting* the object, so that it will no longer be part of the set of selected objects.

5.2.1.3 Move Actions

The Select tool is also used to move objects on the UrduKashishStyler drawing area. To move an object, user points to it, holds down the left mouse button, and drags the mouse to position the object in a new location. When the mouse button is pressed, the bounding rectangle of the object appears. As the mouse is dragged, the shape of object in form of dotted line is traced out to show object moves with the mouse. The user then releases the mouse when the shape is at the desired position. The object then appears at the new location.

To make very small movements, it is often better not to try to drag the object to a precise position with the mouse. Instead, use the keyboard's cursor control (or 'arrow') keys to carry out small moves. The up-arrow key moves the selection one pixel up on the scene. Similarly, the left-arrow, right-arrow, and down-arrow keys move the selected object or objects one pixel left, right, or down, respectively.

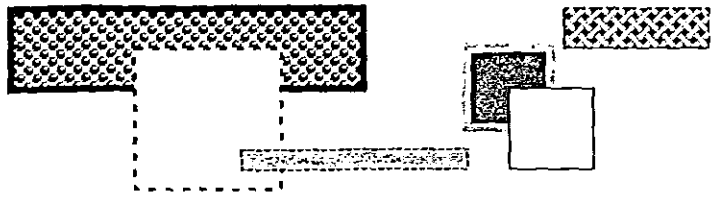


Fig. 5-3 Shapes Drawn with the Rectangle Tool

To draw a rectangle, the user must place the mouse pointer at the point where one corner of the rectangle should be. Pressing and holding down the left mouse button, the user then drags the pointer to the point in the graphical view where the diagonally opposite corner of the rectangle should be. As the mouse is dragged, an outline rectangle is dragged out between the original down-click point and the current mouse location. When the rectangle is the correct size, the user releases the mouse button. At this time, the empty 'feedback' rectangle is replaced by a rectangle with the previously chosen pen style, pen color, fill color and fill pattern features.

The action of drawing with the Rectangle tool can be undone using the Edit menu's Undo command.

5.2.3 The Ellipse Tool

The Ellipse tool is used to draw ellipses. To draw an ellipse, the user must place the mouse pointer at the point where one corner of the ellipse should be. Pressing and holding down the left mouse button, the user then drags the pointer to the point in the graphical view where the diagonally opposite corner of the ellipse should be. As the mouse is dragged, an outline ellipse is dragged out between the original down-click point and the current mouse location. This temporary ellipse, which is continuously reshaped as the mouse is moved, is drawn with a 1-pixel wide black pen. When the mouse button is released, the temporary ellipse is replaced with an ellipse in the currently selected pen style, pattern, pen color, and fill color. In order to draw an ellipse, the mouse must be dragged at least one pixel. If the mouse is released at the same point at which it went down, no graphic is created.

5.2.6 The Free Hand Tool

The Free Hand tool is used for free hand drawing. For free hand drawing, the user must first move the mouse to place the crosshairs at the point on the scene where the drawing is to start. Then the left mouse button must be pressed and held down while the mouse pointer is dragged to the point where free hand is to be drawn. A line is drawn with the mouse movement in the most recently chosen pen style and pen color.

In order to draw a free hand drawing, the mouse must be dragged at least one pixel. If the mouse is pressed and then released at the same point on a scene, no free hand line graphic will be created.

The Edit menu's Undo command can be used to undo the action of drawing freehand drawing with the tool.

5.2.7 The Text Tool

The text tool is used, together with the computer keyboard, to create text graphics. User clicks with this pointer where the bottom right edge of the text should begin. A text editor will appears at that point. The user then types the desired text using the keyboard. UrduKashishStyler can write only one line of text at a time. Tabs are ignored during text entry. Before the text entry action has been completed, the text can be edited. The backspace key can be used to delete the last character.

There are two ways to terminate the creation of a text graphic:

- Typing the Enter key.
- Clicking the OK button with the mouse.

As soon as any of these actions is taken, the text appears with the previously chosen pen style, pen color, fill color and fill pattern features

5.2.8 The Zero Layer Text Tool

The zero layer text tool is used to write text with no outline.When the user selects zero layer text tool UrduKashishStyler Zero Layer Sample dialog appears. After selecting the sample when OK button is pressed, text editor comes in which text can be written. The selected sample applies to the text.

Samples can be edited and can be added to sample gallery for future use. As these samples have no layer so only fill style of the samples can be changed. The samples added are stored permanently unless deleted or edited.

5.2.9 The One Layer Text Tool

The one layer text tool is used to write text with one outline. When the user selects one layer text tool UrduKashishStyler One Layer Sample Gallery appears. After selecting the sample when OK button is pressed, text editor comes in which text is written. The selected sample applies to the text.

Samples can be edited and can be added to sample gallery for future use. The user can edit the fill properties and the outline properties of the sample. The samples added are stored permanently unless deleted or edited.

5.2.10 The Two Layer Text Tool

The two layer text tool is used to write text with two outlines. Two Layer Sample Gallery appears when two layer text tool is selected. After selecting the sample when OK button is pressed, text editor comes in which text is written. The selected sample applies to the text.

Samples can be edited and can be added to sample gallery for future use. The user can edit the fill properties and the outline properties of the first layer and second layer of the sample. The samples added are stored permanently unless deleted or edited.

5.2.11 The Three Layer Text Tool

The three layer text tool writes Urdu text with three outlines. Three Layer Sample Gallery appears when three layer text tool is selected. When OK button is pressed after selecting the desired sample, a text editor appears in which text is written. The selected sample applies to the text.

Samples can be edited and can be added to sample gallery for future use. The user can edit the fill properties and the outline properties of the first layer, second layer and the third layer of the sample. The samples added are stored permanently unless deleted or edited.

5.2.12 Break Text Tool

This tool breaks the sentence to individual words so that each word can be moved independently. When user clicks on any word after selecting break text tool a selection rectangle appears for each word.

As the mouse is dragged by keeping its pointer in any of the selection rectangle, the associated word in form of dotted line is traced out to provide feedback about the scaling operation being carried out. After the mouse is release the word then appears at the new location. Break text operation is undoable.

Some other available features are.

5.3 Undo

Undo command undoes the last undoable action that was taken. UrduKashishStyler has multilevel undo. That means that carrying out Undo will undo the last undoable action taken; executing Undo again will undo the immediately previous action; the next Undo will undo the action before that one; and so on. The number of actions that can be undone is determined by the size of available memory only.

5.4 Redo

The Redo command is used to restore a document to its state before the just-carried-out Undo command. Like Undo, UrduKashishStyler provides a multilevel Redo. For as many times as an author has carried out Undo, that many times can Redo be carried out. If an action which is not undoable takes place after an Undo, Redo ignores that non-undoable action and applies to the earlier Undo action.

The effect of an Undo after a Redo is to undo the command just redone, so Redo is an undoable command.

5.5 Cut

The Cut command in a graphic editor applies to the current selection, which may be one or more graphic objects on the screen. So long as one or more graphic elements or objects are selected, this command is enabled.

The effect of the Cut command is to delete the selection and to place a copy of it on the UrduKashishStyler application's clipboard. The clipboard is a storage area in which copies of data elements are preserved for pasting into appropriate contexts.

A Cut command can be undone. Only the deletion is actually undone. The clipboard will continue to hold its copy of the cut element(s), which can then be pasted.

5.6 Copy

The Copy command in a graphic editor applies to the current selection, which may be one or more graphics which have been selected on that screen. The effect of the command is to place a copy of the selection on the application's clipboard. The clipboard is a storage area in which copies of data elements are preserved for pasting into appropriate contexts.

A Copy command cannot be undone. Undo applies only to actions that change the data that is stored with a document. The clipboard is not stored when a Save action is carried out.

5.7 Paste

The Paste command places a copy of the clipboard's contents into the graphic editor view. For example, user can make copies of one or more graphic objects by selecting in any graphical view and issuing a Copy command.

The newly pasted graphic element or elements will constitute the current selection when the Paste has been carried out.

The Paste command can be undone. Undoing a Paste has no effect on the contents of the clipboard.

5.8 Delete

The Delete command removes the currently selected graphic or graphics from the graphical view that issued the command. After the selection disappears, there is no current selection on the scene. The scene is still active, however, and is therefore the target for a subsequent Paste command. A deletion can be undone with the Undo command.

5.9 Duplicate

In many respects, the Duplicate command works like an immediate copy and paste of the graphic view's current selection. However, it does not change the contents of the clipboard. A user can therefore copy object A, duplicate object B and then paste a copy of A.

After a Duplicate command is carried out, the new graphic element or elements created will be selected and will be highlighted by selection rectangles. The Duplicate command is undoable.

5.10 Select All

The Select All command has the effect of selecting all the objects in the graphic editor. The Select All command is not undoable. If there are no prior undoable actions on the

history list, the Undo command will have no effect on the selection. If there are prior undoable actions, then choosing Undo after Select All will have the effect of undoing the last undoable action. As a side effect, that undoing will change the selection to what it had been just before the undone action was originally carried out.

5.11 Copy Attributes From

This command copies all the attributes of a object to another object. The user selects an object then selects copy attributes from edit menu then click on another object. All attributes of second object are copied to first object. The attributes which are copied includes outline style, color, fill style and fill colors. Copy attributes from is undoable.

5.12 Group

The group command is used to combine several objects into a grouped object. A group is set of objects that behave as one unit. Operations performed on a group apply equally to each of its objects.

When individual graphic elements are combined, the group can be moved and scaled as an entity. Such transformations are automatically applied appropriately to the members of the group.

To carry out the Group command, two or more graphics must first be selected in a graphical editor. (Multiple items can be selected using the select tool either by dragging out a bounding rectangle to enclose the items or by using shift-click.). The Group command can then be issued. Two or more groups can further be grouped and this process can continue to any level of grouping.

The group command is undoable. After the Undo is carried out, the former members of the group will still be selected, as they all were before undo.

5.13 Ungroup

The Ungroup command can be used to ungroup grouped objects. Ungrouping takes effect only at the top level of a group. If one ungroups a group made up of groups which are themselves made up of other groups, for example, only the top group itself is ungrouped. Of course, one can then ungroup its component groups by issuing other ungroup commands.

After the Ungroup command, the former members of the disbanded group are all selected. Immediately subsequent operations will apply to all those objects.

The Ungroup command is undoable. After the ungroup action is undone, the grouped object will be the selected object; just as it was just before the Ungroup command was carried out.

5.14 Order Of Shapes

The objects can be thought of as occupying thin layers, with each object having its own layer. The first object drawn on a scene has the lowest layer. The second object drawn is in a layer just above the first. The third object drawn is in a layer just above the second, and so on. For example, in the figure 5.4, the tall rectangle was drawn first, then the circle, and then the wide rectangle. When graphics overlap each other, objects in lower layers will be partially or completely obscured by objects in higher layers.

UrduKashishStyler provides six commands for moving these layers backward (toward the deepest layer) or forward (toward the top layer). These commands are forward one, back one, to front, to back, in front of and in back of.

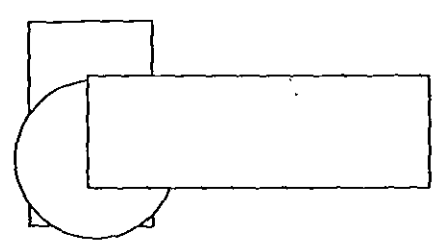


Figure 5.4 Objects overlapping each other

5.14.1 Forward One

The effect of the Move Forward command is to move the layer with the selected object one layer closer to the top. This may result in the object occluding all or part of some other object that formerly occluded it. In the figure 5.5, the tall rectangle was selected. (Note that its lower right selection handle shows even though that corner is occluded by objects in higher layers.) When the Move Forward command was issued, the rectangle's layer moved closer to the top, so that the rectangle partially obscured the circle.

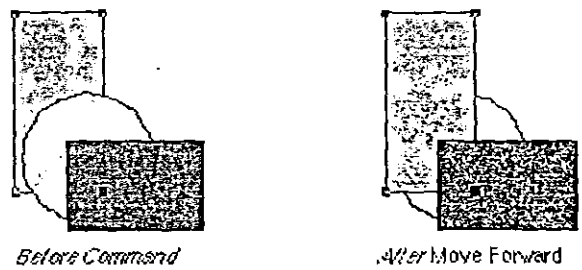


Figure.5.5 The Effect of the Forward One Command

It is not always the case that a Forward one command will make an obscured object move in front of the object that obscures it. If the circle in the figure above were located far off to the right of the two rectangles, then applying the Move Forward command to the tall rectangle would make its layer move above the layer with the circle, but there would be no visual effect until one of the graphics is dragged so that they overlap. When user wants to make one object move in front of another, it will be necessary to issue as many Forward one commands as there are layers separating the objects.

If the selected object is topmost in the graphical view, the Move Forward command has no effect.

If several objects are the target of the Move Forward command, then each of their layers will move one layer closer to the top. The Move Forward command is undoable.

5.14.2 Back One

The Back One command moves the layer of the selected object one step deeper in the stack of layers. This may result in the object being completely or partially occluded by some

other object that it formerly occluded. In the figure 5.6 at the left below, the wide rectangle was selected. When the Back One command was then issued, the circle partially obscured the wide rectangle, as shown at the right in the figure.

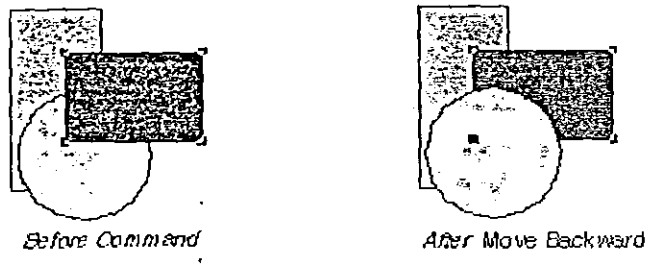


Figure. 5.6 The Effect of the Back One Command

Using the Back One command on an object will not always make it 'go behind' an object that it overlaps. If a number of layers separate the two graphics, it may be necessary to issue the Back One command several times to achieve this effect.

If the selected object is already in the deepest layer of a graphical view, then the Back One command has no effect.

If several objects are the target of the Back One command, then each of their layers moves one step closer to the bottom of the stack of layers. The Back One command is undoable.

5.14.3 To Front

The To Front command moves the layer of the selected object to be the top layer in the graphical view. Any overlapping objects may be partially or completely obscured by the action.

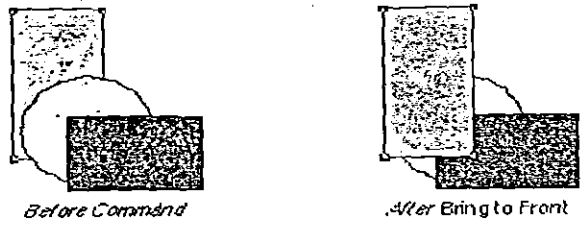


Figure 5.7 The Effect of the Bring to Front Command

If the selected object is topmost in the graphical view, the Bring to Front command has no effect.

If several objects are the target of the Bring to Front command, then their layers become the top layers, stacked in the same order with respect to each other that they were in originally.

The Bring to Front command is undoable. Even if a number of widely separated layers are brought to the front in a single Bring to Front command, the Undo command will restore each layer to its former place in the stack.

5.14.4 To Back

The to Back command moves the layer of the selected object to be the bottom layer in the graphical view. The selected objects may be partially or completely obscured by the To Back action when their layers are sent behind other objects in the view. The figure 5.8 shows the effect of applying the to Back Command to an object that overlaps other objects in a view. Note that the selection handles of a selected object show through overlaid graphics.

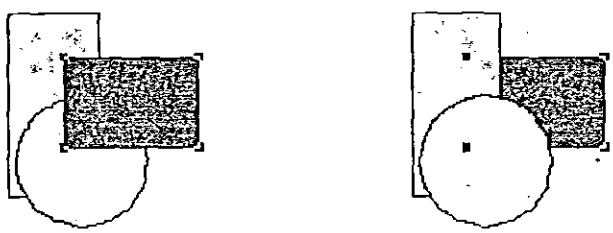


Figure 5.8 The Effect of the to Back Command

If the selected object is deepest in the graphical view, To Back command has no effect. If several objects are the target of the To Back command, then their layers become the bottom layers, stacked in the same order with respect to each other that they were in before the command applied.

The To Back command is undoable. Even if a number of widely separated layers are sent to the back in a single To Back command, the Undo command will restore each layer to its former place in the stack.

5.14.5 In Front Of

The effect of the In Front of command is to move the selected object at top of object clicked after the command. The object may go back and it may come in front depending upon its previous position in the stack of layers. This may result in the object occluding all or part of some other object that formerly occluded it or vise versa. Objects can not be moved relative to itself. The In Front Of command is undoable.

5.14.6 Behind

The behind command moves the selected object behind the target object. The object may go back and it may come in front depending upon its previous position in the stack of layers. This may result in the object occluding all or part of some other object that formerly occluded it or vise versa. Objects can not be moved relative to itself. The Behind command is undoable.

5.15 Transformations

UrduKashishStyler provides two types of transformations.

- Translation
- Scaling

5.15.1 Translation

A translation is applied to an object by repositioning it along a straight-line path from one coordinate location to another. We translate a two-dimensional point by adding translation distances, t_x and t_y, to the original coordinate position (x , y) to move the point to a new position (x', y') (Equation 5.1).

$$x' = x + t_x, \qquad y' = y + t_y \tag{5.1}$$

The translation distance pair (t_x, t_y) is called a translation vector or shift vector. We can express the translation equation 5.1 as a single matrix equation by using column vectors to present coordinate positions and the translation vector:

$$p = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} , \quad p' = \begin{bmatrix} x'_1 \\ x'_2 \end{bmatrix} , \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \qquad (5.2)$$

This allows us to write the two-dimensional translation equations in the matrix form:

$$p' = p + T \qquad (5.3)$$

Translation is a rigid-body transformation that moves objects without deformation. That is, every point on the object is translated by the same amount. A straight line segment is translated by applying the transformation equation 5.3 to each of the line endpoints and redrawing the line between the new endpoint positions. Polygons are translated by adding the translation vector to the coordinate position of each vertex and regenerating the polygon using the new set of vertex coordinates and the current attribute settings.

Similar methods are used to translate curved objects. To change the position of a circle or ellipse, we translate the center coordinates and redraw the figure in the new location. We translate other curves (for example, splines) by displacing the coordinate positions defining the objects, then we reconstruct the curve paths using translated coordinate points.

5.15.2 Scaling

A scaling transformation alters the size of an object. This operation can be carried out for polygons by multiplying the coordinate values (x, y) of each vertex by scaling factors S_x and S_y to produce the transformed coordinates (x', y')

$$x' = x.\, S_x \qquad y' = y.S_y x' \qquad (5.4)$$

Scaling factors s_x scales objects in the x direction, while s_y scales in the y direction. The transformation equation 5.4 can also be written in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.5}$$

$$p' = S.p \tag{5.6}$$

Where S is the 2 by 2 scaling matrix in Equation 5.5.

Any positive numeric values can be assigned to the scaling factors S_x and S_y. Values less than 1 reduce the size of objects; values greater than 1 produce an enlargement. Specifying a value 1 for both S_x and S_y leaves the size of objects unchanged. When S_x and S_y are assigned the same value, a uniform scaling is produced that contains relative object proportions. Unequal values for S_x and S_y result in a differential scaling.

Objects transformed with equation 5.5 are both scaled and repositioned. Scaling factors with values less than 1 move objects closer to the coordinate positions farther from the origin, while values greater than 1 move coordinate positions farther from the origin.

We can control the location of a scaled object by choosing a position, called the fixed point, that is to remain unchanged after the scaling transformations. Coordinates for the fixed point (x_f, y_f) can be chosen as one of the vertices, the object centroid, or any other position. For a vertex with cooridinates (x,y), the scaled coordinates (x', y') are calculated as

$$x' = x_f + (x - S_f)S_x, \qquad y' = y_f + (y - y_f)S_y \tag{5.7}$$

We can rewrite these scaling transformations to separate the multiplicative and additive terms:

$$x' = x.S_x + x_f(1 - S_x) \tag{5.8}$$

$$y' = y.S_y + y_f (1 - S_y)$$

where the additive terms $x_f (1 - S_x)$ and $y_f (1 - S_y)$ are constants for all points in the object.

Polygons are scaled by applying transformations 5.8 to each vertex and then regenerating the polygon using the transformed vertices. Other objects are scaled by applying the scaling transformation equations to the parameters defining the objects. An ellipse in standard position is resized by scaling the semimajor and semiminor axes and redrawing the ellipse about the designated center coordinates. Uniform scaling of a circle is done by simply adjusting the radius. Then we redisplay the circle about the center coordinates using the transformed radius.

5.16 Inside-Outside Tests

When a user clicks on a shape certain tests are required to be performed to check whether the point is inside the shape or outside the shape. Different types of shapes have different types of test.

5.16.1 Circle

The equation of circle can be used to find that whether a point lies inside a circle, on the circle or outside the circle. Equation of circle is

$$Fc (x, y) = x^2 + y^2 - r^2 \qquad (5.9)$$

Any point (x, y) on the boundry of the circle with radius r satisfies the equation Fc(x, y) = 0. If the point is in the interior of the circle, the circle function is negative. And if the point is outside the circle, the circle function is positive. To summarize, the relative position of any point (x, y) can be determined by checking the sign of the circle function.

$$Fc (x, y) \begin{cases} < 0 & \text{if (x, y) is inside the circle boundary} \\ = 0 & \text{if (x, y) is on the circle boundary} \\ > 0 & \text{if (x, y) is outside the circle boundary} \end{cases} \qquad (5.10)$$

5.16.2 Line

As a line has only two points. To check that a point lies on the line or not we use distance formula. The distance between two points P1(x_1, y_1) and P2(x_2, y_2) can be find

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{5.11}$$

Let P1(x_1, y_1) and P_2(x_2, y_2) are the end points of a line. And let the user clicks on a point P3(x_3, y_3). Then if the distance between P1 and P2 is the same as the sum of the distances between P1 and P3 plus P2 and P3 then the point P3 is on the line other wise not.

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} = \sqrt{(x_2 - x_3) + (y_2 - y_3)^2} + \sqrt{(x_1 - x_3)^2 + (y_1 - y_2)^2} \tag{5.12}$$

5.16.3 Text

As the outline of text is a polygon. So we can use the rules for inside outside test of polygons for the text.

Consider a polygon made up of N vertices (x_i,y_i) where i ranges from 0 to N-1. The last vertex (x_N,y_N) is assumed to be the same as the first vertex (x_0,y_0), that is, the polygon is closed. To determine the status of a point (x_p,y_p) consider a horizontal ray emanating from (x_p,y_p) and to the right. If the number of times this ray intersects the line segments making up the polygon is even then the point is outside the polygon. Whereas if the number of intersections is odd then the point (x_p,y_p) lies inside the polygon. The following shows the ray for some sample points and should make the technique clear.

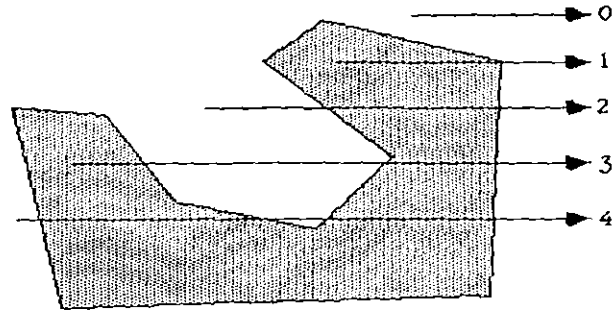


Figure 5.9 Inside Outside Test for Polygon.

For the purposes of this discussion 0 will be considered even, the test for even or odd will be based on modulus 2, that is, if the number of intersections modulus 2 is 0 then the number is even, if it is 1 then it is odd.

The only trick is what happens in the special cases when an edge or vertex of the polygon lies on the ray from (x_p,y_p). The possible situations are illustrated below in figure 5.10.

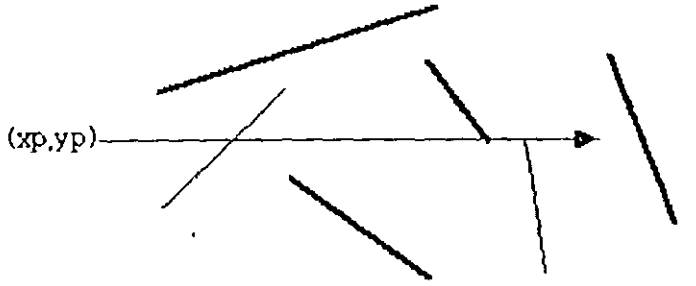


Figure 5.10 Edge or Vertex on the Ray.

The thick lines above are not considered as valid intersections, the thin lines do count as intersections. Ignoring the case of an edge lying along the ray or an edge ending on the ray ensures that the endpoints are only counted once.

Note that this algorithm also works for polygons with holes as illustrated below

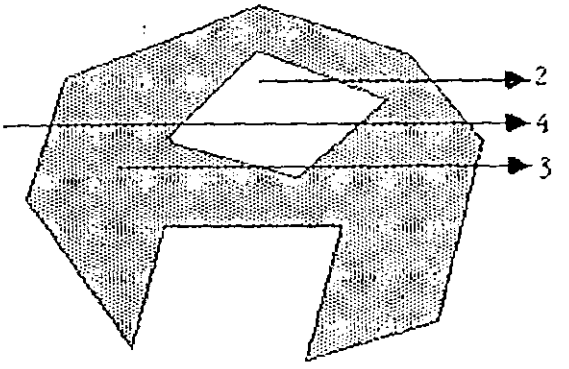


Figure 5.11 Polygon with hole.

The following C function returns INSIDE or OUTSIDE indicating the status of a point P with respect to a polygon with N points.

```
#define MIN(x,y) (x < y ? x : y)
#define MAX(x,y) (x > y ? x : y)
#define INSIDE 0
#define OUTSIDE 1

typedef struct {
   double x,y;
} Point;

int InsidePolygon(Point *polygon,int N,Point p)
{
  int counter = 0;
  int i;
  double xinters;
  Point p1,p2;

  p1 = polygon[0];
  for (i=1;i<=N;i++) {
    p2 = polygon[i % N];
    if (p.y > MIN(p1.y,p2.y)) {
      if (p.y <= MAX(p1.y,p2.y)) {
        if (p.x <= MAX(p1.x,p2.x)) {
          if (p1.y != p2.y) {
            xinters = (p.y-p1.y)*(p2.x-p1.x)/(p2.y-p1.y)+p1.x;
            if (p1.x == p2.x || p.x <= xinters)
              counter++;
          }
        }
      }
    }
    p1 = p2;
  }

  if (counter % 2 == 0)
    return(OUTSIDE);
```

```
  else
    return(INSIDE);
}
```

Another code for determining whether or not a point (x,y) lies inside or outside a polygon is give below this code returns 1 for interior points and 0 for exterior points.

```
int pnpoly(int npol, float *xp, float *yp, float x, float y)
{
  int i, j, c = 0;
  for (i = 0, j = npol-1; i < npol; j = i++) {
    if ((((yp[i] <= y) && (y < yp[j])) ||
         ((yp[j] <= y) && (y < yp[i]))) &&
        (x < (xp[j] - xp[i]) * (y - yp[i]) / (yp[j] - yp[i]) + xp[i]))
      c = !c;
  }
  return c;
}
```

5.17 Getting the Glyph Outline

We have used true type fonts in UrduKashishStyler. TrueType is a common vector font standard used by the Microsoft Windows and Apple operating systems, among others. In a vector font, a series of coordinates define a character's contour, so simple scaling transformations effectively shrink or enlarge the character. Multiplying all the coordinates by two doubles the character's size, for example, and it looks just as good at both resolutions. Operating systems typically allow users to access TrueType font handling without having to know all the details. But font manipulations beyond those supplied by the operating system require a deeper understanding of the TrueType format. Understanding the format, and having the coordinates to each character's contour, opens the door to a world of special text effects like gradient-filling the character's interior, extruding it, placing it realistically on a sphere, and so on.

Microsoft Windows furnishes direct access to TrueType coordinates through the GetGlyphOutline API. GetGlyphOutline supplies the vector points for straight lines and Bezier curves in an abstract coordinate system. Rendering the character then requires deciphering the vector points and drawing the lines and curves with MoveTos and LineTos. The Bezier curves in particular must be decomposed into straight lines and patched together end-to-end to produce the final smooth contour. We will explain the mechanics of drawing a TrueType font.

5.17.1 The GetGlyphOutline API

The signature of GetGlyphOutline is

```
DWORD GetGlyphOutline(

  HDC hdc,              // handle of device context
  UINT uChar,           // character to query
  UINT uFormat,         // format of data to return
  LPGLYPHMETRICS lpgm,  // pointer to metrics struct
  DWORD cbBuffer,       // size of buffer for data
  LPVOID lpvBuffer,     // address of buffer for data
  CONST MAT2 *lpmat2    // pointer to transform matrix
);
```

The handle to device context, hdc, must be valid at the time GetGlyphOutline is called, and it must have the TrueType font of interest selected into it. uChar is the character being interrogated for an outline. uFormat determines whether the data returned is in bitmap (GGO_BITMAP) or vector (GGO_NATIVE) form, the latter being appropriate here. GetGlyphOutline fills in the fields of the GLYPHMETRICS structure pointed to by lpgm with information about the glyph's size and placement; fields gmBlackBoxX and gmBlackBoxY, for example, hold the size of the glyph's bounding box. (See MSDN for a description of the GLYPHMETRICS structure.)

Using GetGlyphOutline always requires two calls. In the first call, parameter cbBuffer is set to 0 and lpvBuffer is set to NULL. This tells GetGlyphOutline to return the size of the buffer needed to hold the glyph data. After the program has allocated a buffer of that size, it calls GetGlyphOutline again, passing the buffer size in cbBuffer and the buffer address in lpvBuffer. When called with these argument values, GetGlyphOutline copies the vector data into the buffer.

Parameter lpmat2 is a pointer to a transformation matrix, which GetGlyphOutline will apply to all points in the glyph before writing them to the buffer. The transformation is applied through matrix multiplication, thus making GetGlyphOutline capable of linear effects such as shearing and rotating..

GetGlyphOutline returns numbers in a fixed point format, in which two integers (fract, value) represent a real number. value represents the part of the real number to the left of the decimal point; fract represents the part to the right of the decimal point, considered as a fraction of 65536. For example, 0.5 becomes (fract, value) = (32768, 0); 2.25 is equivalent to (fract, value) = (16384, 2); and so on. Numbers of this format are stored in structures of type FIXED. The same structure must be used for matrix entries as well.

5.17.2 Polyline and QSpline Records

GetGlyphOutline fills the buffer with a sequence of structures describing the glyph. A glyph consists of one or more "contours". Each contour is described by a TTPOLYGONHEADER structure followed by as many TTPOLYCURVE structures as required to describe it. Each TTPOLYCURVE structure can be either a polyline record or a spline record.

The **TTPOLYGONHEADER** structure specifies the starting position and type of a contour in a TrueType character outline.

```
typedef struct _TTPOLYGONHEADER {
  DWORD    cb;
  DWORD    dwType;
  POINTFX pfxStart;
} TTPOLYGONHEADER, *LPTTPOLYGONHEADER;
```

cb specifies the number of bytes required by the TTPOLYGONHEADER structure and TTPOLYCURVE structure or structures required to describe the contour of the character.

dwType specifies the type of character outline returned. Currently, this value must be TT_POLYGON_TYPE.

pfxStart specifies the starting point of the contour in the character outline.

Each TTPOLYGONHEADER structure is followed by one or more TTPOLYCURVE structures. The **TTPOLYCURVE** structure contains information about a curve in the outline of a TrueType character.

```
typedef struct tagTTPOLYCURVE {
  WORD     wType;
  WORD     cpfx;
  POINTFX apfx[1];
} TTPOLYCURVE, *LPTTPOLYCURVE;
```

wType Specifies the type of curve described by the structure. This member can be one of the following values.

Value	Meaning
TT_PRIM_LINE	Curve is a polyline.
TT_PRIM_QSPLINE	Curve is a quadratic Bézier spline.
TT_PRIM_CSPLINE	Curve is a cubic Bézier spline.

cpfx specifies the number of POINTFX structures in the array.
apfx specifies an array of POINTFX structures that define the polyline or Bézier spline.

Two contours make up a capital 'A', for example: one for the outer contour and one for the triangular hole. Each contour consists of one or more curves, a series of connected, intermingled polyline and QSpline records. A polyline is a series of connected straight lines,

while a QSpline record is a series of connected three-point (quadratic) Bezier curves. A contour is closed, ending where it started. Curve data consists of a series of points, which are represented as POINTFX structures consisting of a FIXED x and a FIXED y.

Polyline records consist of a short (2 byte) integer n followed by n points. The last point of the previous record connects by a straight line to the first point, then straight lines connect subsequent points.

QSpline records also consist of a short integer n followed by n points, but only the last point lies on the glyph itself. These points define a connected series of n-1 Bezier curves.

Figure 5.11 shows a quadratic Bezier curve. A quadratic Bezier curve is defined by three points: controls p1 and p3, and handle p2. The curve begins at p1 in the direction of handle p2, eventually veering back towards p3, where it ends. The handle vectors connecting p1 and p3 to p2 in figure 5.11 are construction lines -- they're shown only to illustrate how the curve runs tangent to one of these vectors before breaking off towards the other control point. Although Windows 95 has built-in Bezier drawing support with functions PolyBezier and PolyBezierTo, these functions draw four-point (cubic) Bezier curves, not three-point (quadratic) curves. (Cubic Bezier curves have two handles; quadratic Beziers convert to cubic Beziers by choosing the cubic handles to be two-thirds of the way from the quadratic control points to the quadratic handle.)

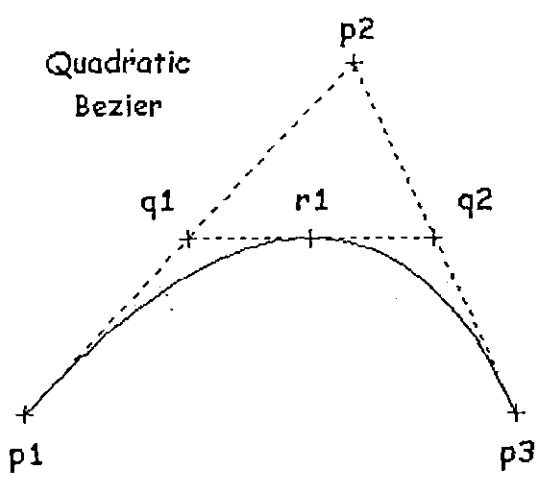


Figure 5.12 Bazier curve with points p1, p2 and p3.

Instead of trying to use the Windows 95 functions, we elected to implement the elegant recursive deCasteljau algorithm, which calculates a series of points along the Bezier curve which are then connected as a polyline. DeCasteljau works by calculating point q1 midway between p1 and p2, and point q2 midway between p2 and p3. Then point r1, the midpoint of segment q1q2, is a point on the curve, and one that partitions the original Bezier curve into left sub-Bezier p1q1r1 and right sub-Bezier r1q2p3. The subdivision process continues recursively to generate as many evenly spaced points on the original Bezier curve as are desired.

TrueType adds an extra twist in the way Bezier curves are stitched together in a single QSpline record. If n = 2 in a QSpline record, there will be a single Bezier with p1 being the last point on the previous record and p2 and p3 the given points. If n = 3, however, there will be two Beziers joined end to end. If we denote the three points by apfx[0], apfx[1], and apfx[2]. The first Bezier curve is defined by:

```
p1 = last point in previous record

p2 = apfx[0]

p3 = (apfx[0] + apfx[1]) / 2
```

The second Bezier curve has points:

```
p1' = p3 on last Bezier

p2' = apfx[1]

p3' = apfx[2]
```

With the exception of the last point, the spline points returned by GetGlyphOutline are the Bezier handles. The curve does not pass through any of the points returned by GetGlyphOutline (since they are handles) except the last point. The control points can be reconstructed from the handles: each control point is the average of two adjacent handles. Since the average of two points is the point exactly midway between them, this ingenious scheme insures that the joined Bezier curves are smooth at the point of juncture. This is because the first Bezier is tangent to the vector connecting p3 to p2, while the second one is

tangent to the vector connecting p1' = p3 to p2'. The two vectors point in diametrically opposite directions, by construction (compare with Figure 5.12).

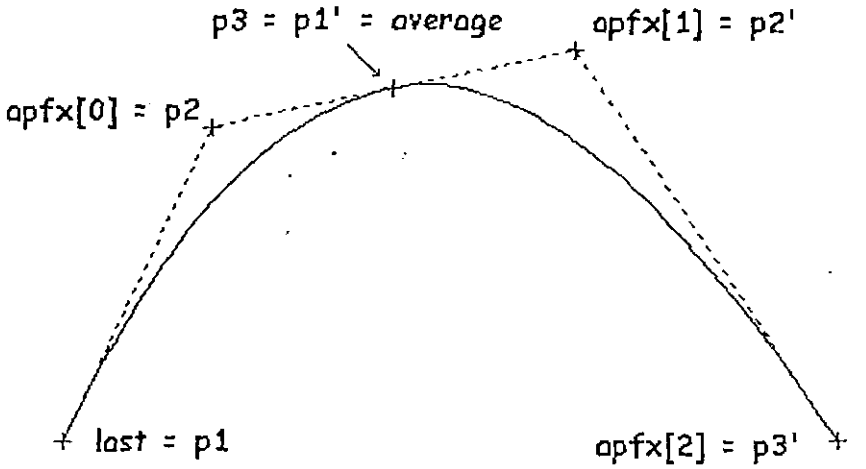


Figure 5.13 Two Quadratic Beziers Joined at p3

The same averaging scheme applies if there are more than two Bezier curves in one QSpline record, they patch together continuously at each juncture, insuring smoothness at any resolution.

5.17.3 Representing an 'A'

Consider the TrueType representation of the Times New Roman capital 'A' in Figure 5.14, for example. First comes the outer contour, pictured here, then a second contour for the hole (not shown). The outer contour's start point, marked by a green cross, is on the right underside of the horizontal arm. The contour begins to trace straight to the left, then turns down and to the left to start down the left foot, proceeding all the way around in a clockwise fashion. The blue crosses mark points in polyline records, the red crosses points in QSpline records.

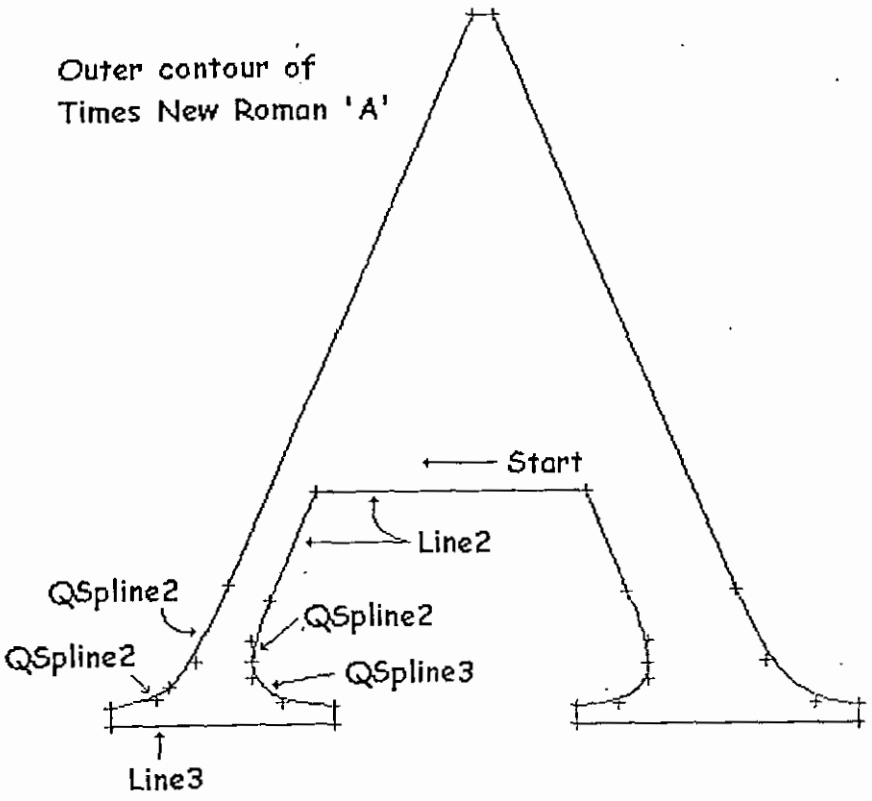


Figure 5.14 Times New Roman A with Polylines and QSplines

If Line2 denotes a polyline record with two points, QSpline3 a QSpline record with three points, and so on, then this contour consists of the following records:

```
Line2
QSpline2
QSpline3
Line3
QSpline2
QSpline2
Line3
QSpline3
Line3
QSpline3
QSpline2
```

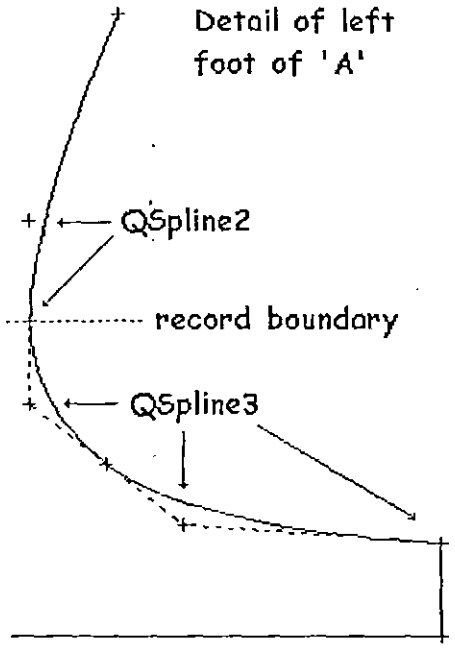


Figure 5.15 Left Foot of A

A detail view of the left foot in Figure 5.15, helps illustrate how the QSplines work. The first QSpline record in the contour begins to define the right edge of the left foot. It has two points (QSpline2) and determines a single Bezier curve. The two points are the handle and second control, the first control being the last point on the previous polyline record. The next QSpline record has three points (QSpline3), where the first two handles are not on the contour and the third is the final control. Compare to Figure 5.12, which shows two quadratic Beziers joined at point p3, essentially the same diagram in a different orientation. Here too the construction lines are drawn, showing that the midpoint of the two interior handles is a control point common to both Beziers.

Chapter 6

Testing and Results

6. Testing

Testing is an important phase during software development life cycle, and shows the stability of the product. Also it helps in comparing the final product with the objectives. Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding.

Testing should focus upon the system's external behavior; a secondary purpose of testing is pushing the limits of the system in order to understand how it fails under certain conditions. A design must be testable. An implementation must be tested.

6.1 Objective of Testing

The overall objective of the testing is to find the maximum number of errors in minimum amount of effort.

6.2 Object Oriented Testing Strategies

Testing begins with unit testing, then progress towards integration testing, and culminates with validation and system testing. In unit testing single modules are tested first. Once each module is tested individually, it is integrated into a program structures while a series of regression tests are run to uncover errors due to interfacing of modules and side effects caused by addition of new units. Finally the system as a whole is tested.

6.3 Types of Testing Done

We conducted various types of testing to make the software stable and error free.

6.3.1 Unit Testing

All the modules of the project were first tested individually by inserting invalid values. Exceptions thrown were properly handled.

6.3.2 Integration Testing

After the modules were tested individually, they were combined to form the final product. All the links and paths were tested. Invalid values were also checked and measure taken to handle them successfully.

Tests of inter object and inter process coordination should be built at several granularity levels. For example, tests of two or three interacting objects, dozens of objects, and thousands of them are all needed.

6.3.3 Black Box Testing

The software was tested for graphical user interface and measures taken that expected output is generated on input.

6.3.4 White Box Testing

Prior testing is part of white box testing in which we look inside the code. Here we can often find errors, Tests that force most or all computation paths to be visited, and especially those that place components near the edges of their operating conditions form classic test strategies.

6.3.5 Beta testing

Use by outsiders rather than developers often makes up for lack of imagination about possible error paths by testers.

6.3.6 System Testing

The software was checked under different operating system like Windows NT and 2000.

6.3.7 Portability testing

Tests should be applied across the range of systems on which the software may execute. Tests may employ suspected non-portable constructions at the compiler, language, tool, operating system, or machine level.

6.3.8 Regression testing

Tests should never be thrown out (unless the tests are wrong). Any changes in classes, etc., should be accompanied by a rerun of tests. Most regression tests begin their lives as bug reports.

6.4 Evaluation

Evaluation of the software is carried out to check the stability and usability of the product being developed .We took measures to ensure that the developed software becomes effective and easy to use. Some of the features of the software are given below.

6.4.1 Efficiency and Effectiveness

The product developed is effective and efficient.

6.4.2 Accuracy

The software provided reliable results. Format which is not supported can not be opened.

6.4.3 Easy to Use Graphical User Interface

The graphical user interface used is simple but not multilingual and steps taken that no problems arise during option finding.

encodings; yet whenever data is passed between different encodings or platforms, that data always runs the risk of corruption.

Unicode provides a unique number for every character. The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystem, Microsoft, Oracle, SAP, Sun, Sybase, Unisys and many others. Unicode is required by modern standards such as XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc., and is the official way to implement ISO/IEC 10646. It is supported in many operating systems, all modern browsers, and many other products.

The emergence of the Unicode Standard, and the availability of tools supporting it, are among the most significant recent global software technology trends. We are planning that our software also use Unicode as encoding system.

7.2.2 Use of OTF fonts

We are using true type font format but the software capabilities can be dynamically enhanced by using OpenType Fonts. OpenType is a new cross-platform font file format developed jointly by Adobe and Microsoft. Adobe has converted the entire Adobe Type Library into this format and now offers thousands of OpenType fonts.

The two main benefits of the OpenType format are its cross-platform compatibility (the same font file works on Macintosh and Windows computers), and its ability to support widely expanded character sets and layout features, which provide richer linguistic support and advanced typographic control.

7.2.3 Using .Net as Development Tool

As Visual C++ 6 does not support OpenType Font and Java also does not fully support OpenType so the solution is using .Net technology which fully support OpenType format and is now considered as ideal tool for development of windows based applications. The functionality of the software can be drastically increased by using .Net as development tool.

7.2.4 Multilingual text designing

Using .Net as development tool and providing support for OpenType will provide most of the foundation work for multilingual text designing and by making small changes in the software we will enable the software to support all the languages of the world provided their fonts are available.

Appendices

A. UrduKashishStyler

UrduKashishStyler is a vector based tool specially use for preparing versatile headlines and text with effects.

A.1 Description

With UrduKashishStyler you can perform these functions:

- Draw Line
- Draw Circle
- Draw Rectangle
- Draw Ellipse
- Draw Free Hand Line
- Break Text
- No Layer Text
- One Layer Text
- Two Layer Text
- Three Layer Text
- Gradient Fill
- Solid Fill
- Texture Fill
- Pattern Fill
- Undo
- Redo
- To Back
- To Front
- In Front Of
- In Back Of

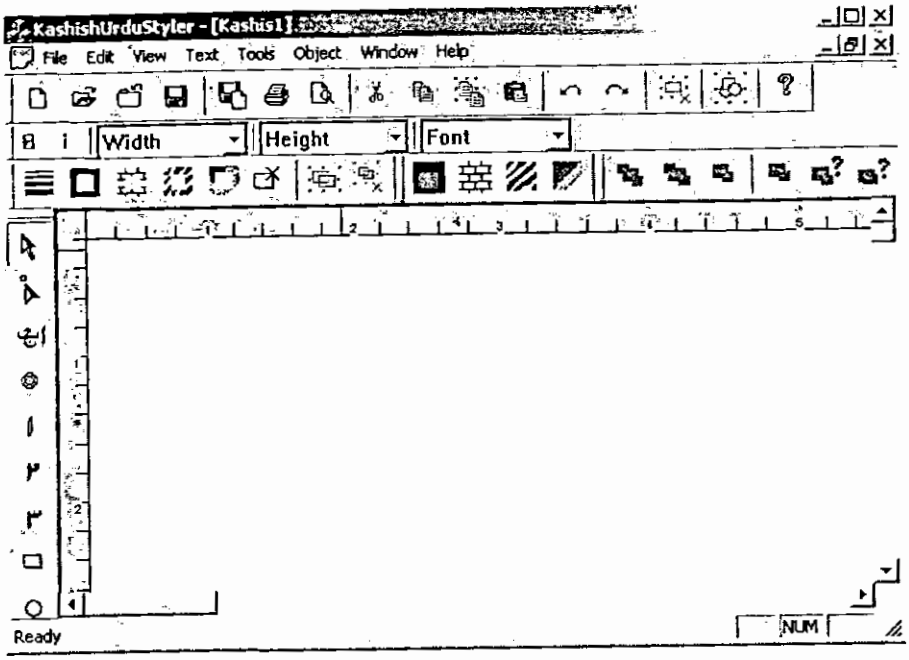


Figure A.1 UrduKashishStyler

- Forward One
- Back One
- Group
- Ungroup

A.2 Command Widget

The Command widget lists a number of sub-menus and commands. They are

- **File**
  - New
  - Open
  - Close
  - Save
  - Save As
  - Print
  - Print Preview

- Print Setup
- Page Setup
- Exit

➢ **Edit**

- Undo
- Redo
- Cut
- Copy
- Paste
- Delete
- Duplicate
- Zoom
- Select All
- Copy Attributes From

➢ **View**

- Toolbar
- Status Bar
- Drawing Bar
- Text Bar
- Kashish Bar

➢ **Text**

- Break Text
- Text
- One Layer Text
- Two Layer Text
- Three Layer Text

- **Tools**
  - Select
  - Circle
  - Ellipse
  - Line
  - Rectangle
  - Free Hand
  - Fill Style
  - Outline Style
- **Object**
  - Order
    - To Front
    - To Back
    - Forward One
    - Back One
    - In Front Of
    - Behind
  - Group
  - Ungroup
- **Window**
  - New Window
  - Cascade
  - Tile
  - Arrange Icons

- **Help**
  - About UrduKashishStyler

B. Glossary

Binary Image (monochrome image) where pixel has only two values generally 0 or 1.

Bitmap font represents each character glyph using a bitmap array and is designed for a specific aspect ratio and character size.

BMP Bitmap image format. An uncompressed image format where the image pixel values are mapped one to one in the image.

Device Context is a data structure that is defined by Windows, which contains information that allows Windows to translate your output request into actions on a particular physical device being used.

Font refers to a complete set of glyphs in a specific typeface, style and weight.

GDI Graphical Device Interface is a class-based application programming interface (API) for C/C++ programmers. It enables to program graphical output independently of the hardware on which it will be displayed.

Glyph is the representation of a character.

Gradient fill is a progression of colors that causes two or more colors to blend from one color to the others smoothly for adding depth and color in the drawing.

Pattern is a simple picture composed of only "on" and "off" pixels.

Pixel slang for picture element. The smallest element of an image. Pixels are arranged in rows and columns to create an image.

Polyline is a series of connected straight lines.

QSpline is a series of connected three-point (quadratic) Bezier curves.

Ramp Fill same as gradient fill.

RGB Red-Green-Blue. An image color space where the image data is represented by the Red, Green, and Blue bit planes of the image.

Scalable font can be resized (enlarged or reduced) without introducing distortion.

Texture is a random, fractally-generated color that is used to give a natural appearance for wood, clouds, stone, ripples, waves, and wrinkles, or create artificial patterns such as checkers, dots, lines, and swirls.

TTF True Type fonts is a font in which each character is defined mathematically as an object, where each object is self-contained, with properties such as color, shape, outline, size and position on the screen.

YCbCr an image color space where the image data is represented by the Luminance and Red and Blue color difference components. Most of the image information is in the Y component.

Zoom process by which an image is magnified by a computer algorithm.

Bibliography and References

Bibliography and References

1. "Computer Graphics"
Donald Hearn and M. Pauline Baker
Prentice-Hall International Inc, 1996

2. "Introduction to Algorithms"
T. H. Cormen, C. E. Leiserson, R. L. Rivest,
The MIT Press, McGraw-Hill, 1990.

3. "http://www.truetype.demon.co.uk"

4. "C/C++ Users Journal, August 1999, TrueType Font Secrets"
Bertrand and Dave Grundgeiger.

5. "Fast Bezier Curves in Windows"
Michael Bertrand
PC Techniques, February/March 1992.

6. "http://www.webopedia.com"

7. "MSDN Library"

Research Paper

ALGORITHM FOR URDU COMPOSITE VECTOR GLYPHS

Muhammad Ali, Khurram Iqbal
Dr. Sikandar Hayat Khiyal
Department of Computer Science, International Islamic University Islamabad.

ABSTRACT

This paper presents the process of making Urdu composite vector glyphs using True Type fonts. A simple and fast algorithm is presented to solve the overlapping problem that arises when Urdu text is written in form of composite vector glyph. The algorithm explicitly copes with the degenerate cases, such that the vector glyph points are input to the algorithm and as a result, the algorithm generates the composite vector glyphs having no overlapping line common in the individual glyphs. The result of this process can be utilized by vector graphics editors to apply special effects on extracted curves.

1. INTRODUCTION

In Urdu many of the characters can have more than one shape according to their position in a word i.e. isolate, initial, middle and final. These shapes are known as glyphs. There are two types of glyphs, bitmapped and vector glyphs. A bitmapped glyph is a collection of bits arranged in rows and columns, designed at a fixed point size for a particular display device, such as a monitor or a printer. These bitmapped glyphs are stored as pictures in the font file. A vector glyph is the glyph in form of outlines and utilizes a vector graphics system to define fonts, where the shape or outline of each character is defined geometrically.

The words in Urdu are constituted with different combination of glyphs provided that initial form always comes first, final in the last and the middle is positioned between the first and last form of glyphs and may vary in number.

In order to make composite vector glyph for Urdu having no overlapping line, we need to get data from True Type font file. The data obtained is in the form of spline and qsplines, then deCasteljau algorithm is applied on qsplines to get vector points. After getting the vector points, proposed algorithm is applied to remove the overlapping line.

2. TRUE TYPE FONTS

In the True Type fonts, each character is defined mathematically as an object, where each object is self-contained, with properties such as color, shape, outline, size and position on the screen. As each object is self contained, transformations may be applied on a large scale while maintaining its original clarity. For this reason True Type is known as vector font format. Probably the greatest thing about storing characters as outline is that only

one outline per character is needed to produce all the sizes of that character. A single outline can be scaled to enormous range of different sizes. This enables the character to be displayed on monitors of different resolutions, and to be printed out at different sizes. To scale a character outline is a simple mathematical operation as are other transformations such as rotation and reflection.

The True Type font resources consist of a sequence of tables that contain the data necessary for drawing the glyphs, measurement information and instructions that the font designer might include. In addition, the font designer can also define additional tables for the outline font resource to support different platforms where outline fonts are available or to provide for future expansion of a font.

3. Extracting Glyph Outline

To access the data from True Type font files data extraction algorithms may be applied. There are utilities available such as GetGlyphOutline[4] Microsoft Windows. It furnishes direct access to TrueType coordinates and supplies the vector points for straight lines and Bezier curves[3] in an abstract coordinate system. Rendering the character then requires deciphering the vector points and drawing the lines and curves. The Bezier curves in particular must be decomposed into straight lines and patched together end-to-end to produce the final smooth contour.

The utility routine fills the buffer with a sequence of structures describing the glyph where each glyph can have one or more contours. A contour is a closed path, ending where it started. For example, the Urdu character 'BAY' is made of two contours, one for the body of BAY and the other for nuqta.

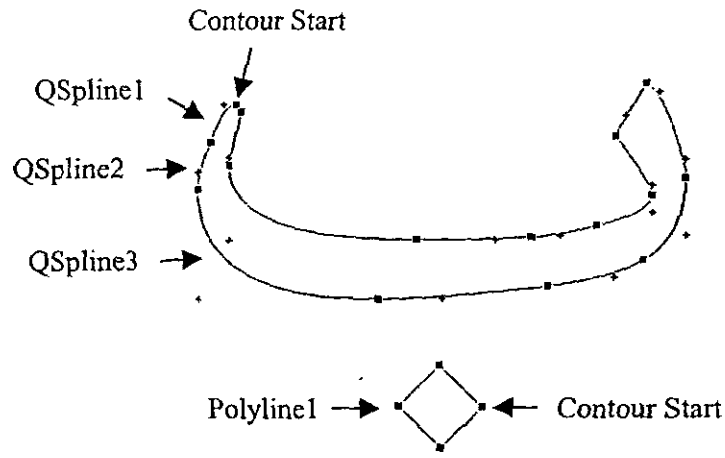


Figure 1: Outline of Urdu character 'BAY' with Polyline and QSpline records

Each contour in the character is described by a TTPOLYGONHEADER[4] structure followed by as many TTPOLYCURVE[4] structures as required to describe it. The

TTPOLYCURVE structure can be either a polyline record or a qspline record where polyline is a series of connected straight lines and qspline record is a series of connected three-point (quadratic) Bezier curves (Figure1).

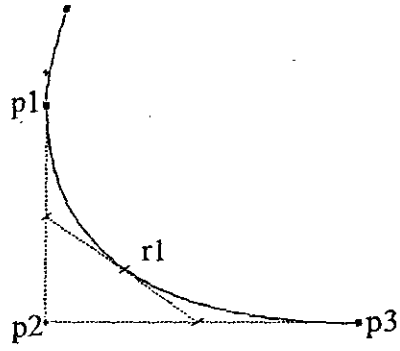


Figure 2: Quadratic Bezier curve determined by control points p1, p3 and handle p2.

A quadratic Bezier curve of QSpline3 (Figure1) is defined by three points, controls p1, p3 and handle p2 as shown in Figure2. The curve begins at p1 in the direction of handle p2, eventually veering back towards p3, where it ends. The handle vectors connecting p1 and p3 to p2 in Figure2 are construction lines, they are shown only to illustrate how the curve runs tangent to one of these vectors before breaking off towards the other control point.

4. Decomposition of Curve by deCasteljau Algorithm

In order to decompose Bezier curve into straight lines we have implemented the elegant recursive deCasteljau algorithm, which calculates a series of points along the Bezier curve. The deCasteljau works by calculating point q1 midway between p1 and p2, and point q2 midway between p2 and p3. Then point r1, the midpoint of segment q1q2, is a point on the curve, and one that partitions the original Bezier curve into left sub-Bezier p1q1r1 and right sub-Bezier r1q2p3 (see Figure 2). The subdivision process continues recursively to generate as many evenly spaced points on the original Bezier curve as desired.

5. FORMING COMPOSITE GLYPH

After applying the deCasteljau algorithm on the glyph outline, we get the points of individual vector glyphs. When these individual vector glyphs are joined to from composite vector glyph an overlapping line is formed between the individual vector glyph. In order to solve this problem we have developed an algorithm that takes individual glyph points as input and the resultant output is a composite vector glyph with no overlapping line.

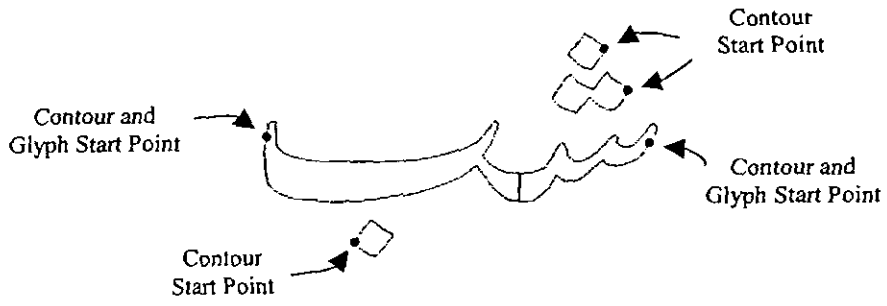


Figure 3: Composite vector glyph with overlapping line.

The steps of algorithm are as follows.

- Suppose A is the array containing all the points in first glyph and B is the array containing the points of second glyph.
- Find out the first intersection point of A and B.
- Copy all the points from A to the resultant array up to the first intersection point.
- The next point of A will be its second intersection point but for B either previous or next point will be the second intersection point.
- If next point of B is second intersection point then copy all the points of B from first intersection point to resultant array until reached at the start of active contour then copy points of B from the start of active contour until reached at the second intersection point.
- If previous point of B is the second intersection point then copy all the points of B from first intersection point up to the end of active contour in the resultant array then copy all points of B from the start, up to the second intersection point.
- Copy all points of A from second intersection point up to the end in the resultant array
- Copy all the points from the start of B up to the starting index of intersecting contour.
- Copy all points from the ending index of intersecting contour up to the end of B.

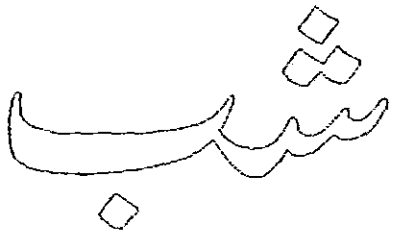


Figure 4: Composite vector glyph without overlapping line

6. CONCLUSION

We may summarize the work done as below.

The process is defined to get data from font file and deciphering the vector points to produce final smooth contour. The algorithm for collecting the resultant vector glyph is introduced, which avoids degeneracy problems of traditional applications.

The proposed algorithm was tested using a program in Visual C++. The algorithm proved to be successful with any combination of Urdu characters. Output of the algorithm can be used by any vector based graphics editing application that uses Arabic script.

REFERENCES

[1] Michael Bertrand and Dave Grundgeiger. "TrueType Font Secrets", C/C++ Users Journal, August 1999.

[2] Michael Bertrand. "Fast Bezier Curves in Windows", PC Techniques, February/March 1992, pp. 25-30. (Reprinted in the book PC Techniques C/C++ Power Tools, 1992, pp. 213-225.)

[3] MSDN Library.

[4] http://www.webopedia.com

[5] http://www.truetype.demon.co.uk