TEXT PROCESSING SYSTEM FOR CULTURAL HERITAGE & MULTIMEDIA APPLICATIONS

Ramil Valitov
Kazan State University, Faculty of Mathematics and Mechanics
Kremlevskya St. 18, Kazan, 420008, Russian Federation
E-Mail: ramilbox@mail.ru
URL: http://www.ksu.ru

Abstract:
Modern applications of Cultural Heritage that use innovative technologies as 3D tours, virtual cultural tours, VR, etc. perform complex rendering of objects and need hi-quality graphic output. The higher level of text layout and visualization complexity leads to problems of rendering the text in general and/or mixed simultaneous display of the text and application’s own intricate graphic data like 3D scenes. The author examined widespread requirements of multimedia applications for text related data rendering, processing and output. He developed a text processing system that meets these requirements, performs fast flexible text rendering with extended graphic capabilities and supports multifarious post-rendering routines.

EXISTING METHODS FOR TEXT RENDERING & PROCESSING

It is quite easy to display a string or a formatted text using a single font and color for all characters. Windows GDI cannot automatically display images within a text and display a more complex text, which may contain characters of different color, font, style or size. Windows GDI+ also does not support such output, although it provides better text rendering features. One of the approaches to this problem is using some ActiveX components or controls like Rich Edit that perform the requested operations. Usually they also support user interaction routines like hit testing, hyperlinks handling, and text editing. Each of these controls is designed to work with one or more predefined file formats (HTML, PDF, etc.). Most of the ActiveX components do not support custom implementation; therefore, they must be inserted into a window form or a dialog box beforehand. Besides, they may also require placing into a .NET or MFC project. Such restrictions prevent these components from using in applications that need custom interface or special graphic output features.

Rich Edit control can be used in any sort of application and supports many useful features: embedded objects, drag and drop editing, Unicode and ANSI (single-byte character set (SBCS) and multibyte character set (MBCS)) editing. The control handles RTF format and can display almost everything using OLE technology. Rich Edit control can work in several modes:
1) Standard mode – the control has its own window that is used to display and edit text;
2) Bottomless mode helps to adjust control’s height according to the data being passed to it;
3) In Windowless mode, the control does not own a window. It displays text into some part of application’s client area.

Rich Edit control is applicable for most projects that use text scrolling, but the control cannot page text. Even it has a bottomless mode, it can properly page only plain
text without images. It also cannot use private font collections, does not support output quality control, etc. The most significant point is that it supports only basic poor design features, which is probably caused by RTF format limitations. More and more modern multimedia applications exactly need special variable different visual effects and unique graphic in general also concerning text output. The last becomes particularly important in multimedia applications in which textual information plays a great role, e.g. multimedia encyclopedias, libraries, dictionaries, educational programs, etc. Extension and expansion of text rendering capabilities can bring freshness, new appearance and interface design to the whole program.

THE TEXT PROCESSING SYSTEM (TPS)

Modern applications of Cultural Heritage that use innovative technologies as 3D tours, virtual cultural tours, VR, etc. perform complex rendering of objects and need high-quality graphic output. Most of such applications and other types of multimedia applications need to display formatted text with embedded images and perform post-rendering processing of the data, e.g. hit testing and hyperlinks handling. The higher level of text layout and visualization complexity leads to problems of rendering the text in general and/or mixed simultaneous display of the text and application’s own intricate graphic data like 3D scenes. The author examined widespread requirements of multimedia applications for text related data rendering, processing and output. He developed a text processing system (TPS) that meets these requirements, performs fast flexible text rendering with extended graphic capabilities and supports multifarious post-rendering routines.

The TPS is a set of C/C++ based classes for text handling. The TPS receives text with optional embedded images and miscellaneous visualization information from the application and returns rendered ready to display bitmaps, which can be used anywhere. It also makes it easy to handle all sorts of rendered data associated tasks by providing graphics related information (position data, coordinates, dimensions, etc.) and performing miscellaneous methods such as hit testing and hyperlinks support. The TPS mathematically calculates all positions, sizes, converts them according to display physical dimensions and graphic resolution, defines word breaks, etc. The final rendered data can be displayed in a standard scrolling mode; however, the TPS has special paging method, which can be interesting for multimedia encyclopedias. It can divide text into separate pages so the document can be displayed as a virtual book. The paging method does not just break the document each time it fits the page’s area, it also tries to arrange embedded images to make them look better in the page’s image. It places document to fit the page without sharpen gaps and breaks between pages.

The application gets graphics and quality control over the rendering process and can execute real-time rendering. The TPS supports private font collections that allow creating a set of user-defined fonts that will be used for text output. These fonts do not have to be installed and registered in the system; they are stored as application’s local data files and do not have an influence on system settings. It can help to avoid unexpected visual errors and inaccuracies of displaying a document induced by missing required fonts installed in a system or inconsistency between existing installed fonts and fonts used to create the document. Applications of course can install the required fonts to the end-user system but they can face with possible problems, e.g. an earlier version of font may have already been installed or a different font with the same name exists in the system. Simple replacement of existing fonts may damage display of user’s private
documents. Private font collections provide an easier, safer and more secure way of using program’s local fonts without altering system’s settings.

The TPS also has a multilevel quality control, which helps to achieve optimized balance between performance time costs and high quality rendering. The control independently adjusts rendering quality of different types of objects. The TPS supports subpixel antialiasing, which gives text and bitmaps a smoother appearance. Text can have five levels of quality; the first four are defined by turning on and off antialiasing and/or hinting. Hinting provides a more clear text output by expressly adjusting each character to fit its intended area. The fifth mode enables ClearType technology, which optimizes text output to look better on an LCD screen. The TPS also controls rendering quality of embedded images by setting scaling methods it uses for drawing bitmaps.

DEVELOPED FILE FORMATS FOR THE TPS

Two extendible HTML-like file formats MCTD and DDPM were specially developed as the TPS oriented data storage formats for mediums. They form entire data description, define imaging parameters and grant complete access to unique features of the TPS.

MCTD (Multiple Content Text Document) is an electronic document file format that describes data to present and its relationship with other files. MCTD represents logically formatted text, which may contain embedded images and hyperlinks that connect the document with other files: audio, video, etc. Design and the way of rendering of the document depend on DDPM (Document Data Presentation Model). It describes graphical depiction of the document: type styles, usable fonts, text layout and visual effects applied to the document. It also determines logical meaning of text parts within the document: it defines tags for titles, body text, quotations, acronyms, abbreviations, etc.

The appearance of MCTD can be easily changed by replacing its DDPM file. Thus, there is a clear partition of the data between the document itself and its presentation model, which only defines logical meaning of text parts and their appearance, but does not have any influence on document’s contents. Both MCTD and DDPM support Unicode and Unicode Surrogates that allow using over a million of characters and displaying multilingual text. Documents are displayed correctly and identically on different operating systems (Windows 98/Me/NT/2000/XP/.NET).

As an example of distinctive graphic feature, DDPM provides seven types of brushes for text painting:

1) Solid Brush;
2) Hatch Brush;
3) Texture Brush;
4) Linear Gradient Brush;
5) Linear Gradient Brush with customizable blend patterns filling;
6) Path Gradient Brush;
7) Path Gradient Brush with customizable blend patterns filling.

Brushes are used to fill text (character’s foreground and background) and document’s background. Each brush has its own unique behavior and filling methods. The set of brushes mentioned above supports:

- Common plain one-color filling;
- Color identification by its predefined name;
- Painting using ready predefined hatched patterns;
- Filling an interior of a shape by an image or its part (Texture Brush) with real-time preprocessing: image zooming, color channels (alpha, red, green, blue) transformations, color conversion into shades of gray;
- Different tiling and transformations of coordinates for some types of brushes;
- Creation and use of brushes with customizable blend patterns filling (Gradient Brushes) that paint a shape with gradually changing colors;
- Complex multicolor path brushes based on desired shape.

Each color is represented by four components: alpha, red, green and blue that make each visible element of the document have an alpha-channel that specifies its transparency.

Each character can be displayed using a separate font, which is identified by its family name (Times New Roman, Arial, etc.). Italic, bold, underlined and strikeout effects or their combinations can also be applied. Text can be left, right or center aligned or justified. Horizontal and vertical indentations can be specified for paragraphs.
An object-oriented approach is applied in DDPM: each element of design (e.g. brush) is considered an object, a unit of some class with its corresponding properties, methods, etc. New objects can be defined based on existing object of the same class. All objects pass through automatic initialization by objects constructor. Thus, a newly generated object contains valid values and it is immediately ready to use.

DDPM supports use of absolute and relative measure units: pixels, points, inches, centimeters, millimeters, pikes and percents. They give flexibility in setting design parameters and allow automatic sizing and positioning for correct document output on different computers based on monitor’s physical dimensions and graphical resolution. The parameters are usually real numbers that if needed are automatically converted into integer values or into some range of values without visible loss of information. A number, string, object name, etc. can be assigned to a parameter, besides its value can be increased or decreased by several percents.

Grouping of similar objects, import and combination of DDPM files is used to reduce file size. Grouping allows setting indicated properties at once for several objects or for several array elements within an object. Owing to import ability, it is possible to create a new DDPM as a copy of another DDPM, add or change its objects without physically modifying the original file.

CONCLUSION

The TPS can be implemented as a static library, dynamic link library or an ActiveX component and is currently realized as a cross-platform version for Microsoft Windows 98/Me/NT/2000/XP/.NET operating systems. It supports multilingual text processing, performs fast hi-quality real-time text related data rendering and post-visualization routines. It is applicable to all kinds of software programs especially multimedia and Cultural Heritage projects that need extended or special graphic capabilities. The prospects are support for easy from right to left text output, enhancement of data elements (e.g. adding tables as a built-in element), development of version for Unix-based operating systems.

REFERENCES