

FrontPanelText サンプル

*このサンプルは、Microsoft ゲーム開発キットのプレビュー (2019 年 11 月) に対応しています。*

# 説明

FrontPanelText サンプルでは、CPU を使用して、Xbox One X Devkit と Project Scarlett の Devkit のフロント パネル ディスプレイにテキストを描画する方法を示します。このサンプルは、.rasterfont ファイルを読み込むことができる RasterFont というクラスを使用します。名前が示しているように、.rasterfont ファイルには、各グリフがラスタライズされた単純なピクセルベースのフォントが格納されています。この形式は CPU 上でレンダリングするのに適しています。RasterFont クラスには、テキストのレンダリングを簡単にする printf スタイルのメソッドが用意されています。PC にインストールされている True Type フォントから独自の .rasterfont ファイルを作成する方法の詳細については、RasterFontGen サンプルも参照してください。

# サンプルのビルド

Xbox One の devkit を使用している場合は、アクティブなソリューション プラットフォームを Gaming.Xbox.XboxOne.x64 に設定します。

Project Scarlett を使用している場合は、アクティブなソリューション プラットフォームを Gaming.Xbox.Scarlett.x64 に設定します。

*詳細については、GDK ドキュメントの* 「サンプルの実行」*を参照してください。*

# サンプルの使用

FrontPanelText サンプルは、フロントパネルを内蔵した Xbox One X Devkit および Project Scarlett Devkit を対象としています。サンプルを開始すると、いくつかのサンプル テキストがフロント パネル ディスプレイにレンダリングされます。フロント パネルの方向パッド (左、右) を使用してテキストのフォント フェイスを変更し、フォント サイズを変更します (上、下)。 方向パッドの上を使用するとフォント サイズが大きくなり、方向パッドの下を使用するとフォント サイズが小さくなります。

方向パネルのボタンを押して (選択)、フロント パネル ディスプレイからバッファーをキャプチャし、タイトル スクラッチ フォルダーにある .dds ファイルに結果を保存することもできます。

次の画像は、異なるサイズでレンダリングされているいくつかのフォント オプションを示すサンプルのスクリーンショットです。



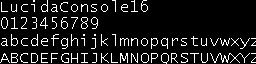
ボタンごとに 1 つの LED

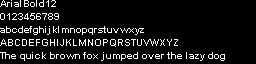
プログラミング可能なボタン 5 個

256 x 64 x 4bpp OLED ディスプレイ

方向パッド + 選択

Lucida Console はフロント パネルで読みやすいフォントです。固定幅フォントであるため、テキストのレイアウト ジオメトリが予想可能であることから、メニューやウィジェットに適しています。また、このように高さ 12 ピクセルのサイズでも読みやすく、5 行のテキストがディスプレイに楽に収まります。

これは、同じフォントを 16 ピクセルのサイズでレンダリングしたものです。このサイズでは、ディスプレイには 4 行のテキストが入ります。

Arial は固定幅ではありません。これは、小文字のアルファベットの幅を大文字と比較するとわかります。Arial は 12 ピクセルの高さでも読みやすく、固定幅のフォントに比べてディスプレイの水平方向に多くの文字を収めることができます。

RasterFont ツールチェーンを使用することで、システムにインストールされている TrueType フォントを使用して、.rasterfont ファイルを生成できます。これは symbol フォントの例です。symbol フォントの文字を使用して、シンプルな UI 要素 (矢印、ボタンなど) を表示できます。 

# 実装に関する注意事項

フロント パネルのテキストは、RasterFont オブジェクトを使用して CPU 上でレンダリングされます。RasterFont オブジェクトを作成するには、.rasterfont ファイルのファイル名をコンストラクターに渡します。次に例を示します。

auto myFont = RasterFont(L"Assets\\LucidaConsole16.rasterfont");

独自のプロジェクトでは、サンプルと共に提供される .rasterfont ファイルを使用できます。RasterFontGen.exe ツールを使用して独自のファイルを作成することもできます。RasterFontGen では、システムにインストールされている任意の True Type フォントから、さまざまなサイズやオプションを使用して .rasterfont ファイルを作成できます。

次のコードは、RasterFont オブジェクトの架空のエンドツーエンドの使用方法を示しています。

// .rasterfont ファイルを読み込む

auto myFont = RasterFont(L"Assets\\LucidaConsole16.rasterfont");

// フロント パネル ディスプレイのバッファー記述子を取得する

BufferDesc fpDesc = m\_frontPanelDisplay->GetBufferDescriptor();

// 書式設定された文字列をバッファーに描画する

myFont.DrawStringFmt(fpDesc, 0, 0,

L"Simple Addition\n%i + %i = %i",

1, 1, (1 + 1));

// バッファーをフロント パネルに表示する

m\_frontPanelDisplay->Present();

BufferDesc は、CPU バッファーの幅と高さを追跡する構造体です。RasterFont はメモリ内の任意のアドレスにテキストをレンダリングできます。必要なのは、バッファーのサイズを記述する BufferDesc だけです。目標とするフロント パネル ディスプレイをより簡単に特定できるように、このサンプルでは、フロント パネルのバッファーを管理する FrontPanelDisplay クラスを使用します。FrontPanelDisplay::GetBufferDescriptor() を使用して、RasterFont を使用してテキストをフロント パネルにレンダリングするのに適した BufferDesc を取得します。

DrawStringFmt を使用してテキストをバッファーに描画します。これは標準のライブラリ関数である printf () と似ています。ここでは、テキストの x 座標と y 座標のほかに BufferDesc も必要です。DrawStringFmt は、テキストのレイアウト時の改行をサポートしています。

RasterFont で用意されているテキスト レンダリング メソッドの概要を次に示します。

// MeastureString および MeasureStringFMt は

// レイアウト目的でのテキスト範囲の計算に役立ちます

RECT MeasureString(const wchar\_t \*text) const;

RECT MeasureStringFmt(const wchar\_t \*format, ...) const;

// バッファーへの基本的なテキストのレンダリング

void DrawString(const struct BufferDesc &destBuffer, unsigned x, unsigned y,

const wchar\_t \*text) const;

// バッファーへの書式設定されたテキストのレンダリング

void DrawStringFmt(const struct BufferDesc &destBuffer, unsigned x, unsigned y,

const wchar\_t \*format, ...) const;

// 次の DrawString のバリエーションには

// さまざまなグレーの色合いを指定するための

// shade パラメーターがあります

void DrawString(const struct BufferDesc &destBuffer, unsigned x, unsigned y,

uint8\_t shade, const wchar\_t \*text) const;

void DrawStringFmt(const struct BufferDesc &destBuffer, unsigned x, unsigned y,

uint8\_t shade, const wchar\_t \*format, ...) const;

// 1 つのグリフを正確に配置するためにグリフ固有のメソッドを使用します

RECT MeasureGlyph(wchar\_t wch) const;

void DrawGlyph(const struct BufferDesc &destBuffer, unsigned x, unsigned y,

wchar\_t wch, uint8\_t shade = 0xFF) const;

RasterFont に関する注意事項:

* shade パラメーターを指定した DrawString() バリエーションは、白地に黒のテキストをレンダリングするために使用できます。たとえば、メニュー システムの一部の行で、白地に黒のテキストを使用する場合などです。DrawString() では背景のピクセルが描画されないことに注意してください。したがって、白地に黒のテキストを表示するには、最初に白い長方形を描く必要があります。必要な長方形の範囲を判別するには、MeasureString メソッドを使用します。
* MeasureGlyph() および DrawGlyph() は、個々のグリフを正確に配置するのに便利です。これらのメソッドではグリフの境界ボックスのみが使用され、隣接するグリフの間隔と、通常の縦書き/横書きのテキストのレイアウトに使用されている垂直オフセットが考慮されません。これにより、グリフを正確に配置できます。たとえば、symbol フォントのグリフを UI 要素またはウィジェットとして使用する場合などです。(MeasureGlyph() および DrawGlyph() のいくつかの例が FrontPanelDemo サンプルにあります。)
* RasterFont で提供される基本的な縦書き/横書きテキストに満足できない場合、基になる RasterGlyphSheet クラスを活用することを検討してください。このクラスには、カスタムの縦書き/横書きテキスト実装の記述に使用できる ForEachGlyph() テンプレートが用意されています。ForEachGlyph() の使用方法の優れた例を、さまざまな RasterFont::DrawString\*() メソッドの実装で確認できます。

その他の実装に関する注意事項:

* Xbox One X Devkit または Project Scarlett Devkit では、::XFrontPanelIsAvailable() は true を返し、完全な API が使用できるようになります。 それ以外の場合、::XFrontPanelIsAvailable() は false を返し、その他の ::XFrontPanel\*() 関数は失敗の HRESULT コードを返します (Xbox One、Xbox One S、物理的なフロント パネルのないメーカーのコンソールなど)。
* すべてのフレームでフロント パネルに表示する必要はありません (::XFrontPanelPresentBuffer())。そうではなく、1 つ以上のピクセルが変更されたときだけ表示する必要があります。したがって、このサンプルには、ディスプレイ バッファーに変更があるたびに設定される m\_dirty メンバーがあります。
* また、変更があったときだけ光の状態を設定する必要があります。
* フロント パネル バッファーに直接アクセスすることはできません。代わりに、独自のバッファーを管理して、バッファーのアドレスを ::XFrontPanelPresentBuffer() に渡す必要があります。Sample::CaptureFrontPanelScreen() では、m\_panelBuffer の内容が単に DDS サーフェスのピクセル ペイロードとして使用されています。

# 更新履歴

2019 年 4 月、サンプルの最初のリリース。

2019 年 11 月、Project Scarlett Devkit のサポート。

# プライバシーに関する声明

サンプルをコンパイルして実行すると、サンプルの使用状況を追跡するため、サンプルの実行可能ファイルのファイル名が Microsoft に送信されます。このデータ収集を無効にするには、「Sample Usage Telemetry」とラベル付けされた Main.cpp 内のコードのブロックを削除します。

Microsoft のプライバシーに関する声明の詳細については、「[Microsoft プライバシー ステートメント](https://privacy.microsoft.com/en-us/privacystatement/)」を参照してください。