

FrontPanelTextサンプル

# *\*このサンプルは Xbox One XDK （2017 年 6月） とで動作可能です。*

# 概要

FrontPanelTextサンプルは、CPUを使用してXbox One X 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ファイルをロードできるRasterFontというクラスを使用しています。名前が示すように、.rasterfontファイルには、各グリフがラスタライズされたシンプルなピクセルベースのフォントが含まれています。このフォーマットはCPU上でのレンダリングに適しています。RasterFontクラスはテキストのレンダリングが簡単であるprintfスタイルのメソッドを提供します。PCにインストールされているTrue Typeフォントから自身の.rasterfontファイルを作成する方法については、RasterFontGenサンプルも参照してください。

# サンプルの使用

FrontPanelTextサンプルは、統合されたフロントパネルを備えたXbox One X DevKitを対象としています。サンプルを開始すると、サンプルテキストがフロントパネルにレンダリングされます。フロントパネルのDPAD（左、右）を使用して、テキストのフォントフェイスとフォントサイズ（上、下）を変更します。DPADの上を使うとフォントサイズが大きくなり、DPADの下を使うとフォントサイズが小さくなります。



ボタンごとに 1 つの LED

5 倍のプログラム可能ボタン

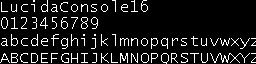
256 x 64 x 4 bpp の OLED ディスプレイ

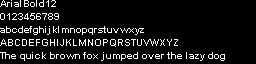
方向パッド + 選択

DPADボタンを押すと（選択）、フロントパネルディスプレイからバッファをキャプチャし、その結果をTitle Scratchフォルダにある.ddsファイルに保存します。

次の画像は、サンプルのスクリーンショットでさまざまなサイズでレンダリングされたいくつかのフォントオプションを示します。

Lucida Consoleはフロントパネル上でかなり読みやすいです。これは固定幅のフォントであり、テキストは予測可能なレイアウトジオメトリを持つことになりますので、メニューやウィジェットに最適です。また、12ピクセルの高さでも読むことができて、このサイズでは5行のテキストがディスプレイに快適に収まります。

16ピクセルのサイズでレンダリングされた同じフォントです。このサイズではディスプレイはテキストの4行を配置することに注意してください。

Arialの小文字のアルファベットの幅と大文字のアルファベットの幅を比較すると、固定幅ではないことが分かります。Arialは12ピクセルの高さで読むことができて、固定幅フォントと比較してディスプレイ上により多くのテキストを水平方向に収めることができます。

RasterFontツールチェーンを使用すると、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。これはシンボルフォントの例です。シンボルフォントの文字は、単純なUI要素（例えば、矢印、ボタンなど）のレンダリングに使用できます。 

# 実装上の注意

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上に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に渡します。例えば：

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

ご自分のプロジェクトの場合は、サンプルに付属の.rasterfontファイルを使用することも、RasterFontGen.exeツールを使用して自分のプロジェクトを作成することもできます。RasterFontGenはシステムにインストールされた任意のTrue Typeフォントから、さまざまなサイズとオプションで.rasterfontファイルを作成することを可能にします。

次のコードは、RasterFontオブジェクトの仮想的なエンドツーエンドの使用方法を示しています。

// .rasterfontファイルをロードします

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ディスクリプタを取得します

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

//フォーマットされた文字列をバッファに描画します

myFont.DrawStringFmt(fpDesc, 0, 0,

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

1, 1, (1 + 1));

//フロントパネルにバッファを表示する

m\_frontPanelDisplay->Present();

BufferDesc は、CPU バッファ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することができます。必要なのは バッファの大きさを記述するBufferDesc のみです。フロントパネルディスプレイを簡単に対象とするため、このサンプルでは フロントパネル用のバッファを管理する FrontPanelDisplay クラスを使用しています。FrontPanelDisplay::GetBufferDescriptor（） を使用して、RasterFont を使用してフロントパネルにテキストをレンダリングするのに適している BufferDesc を取得します。

DrawStringFmt はテキストをバッファに描画するために使用されます。これは標準ライブラリ関数 printf（） に似ています、 。BufferDesc テキストのx座標とy座標が必要であることに注意してください 。DrawStringFmt は、テキストをレイアウトするときの改行をサポートします。

RasterFont が提供するテキストレンダリングメソッドの要約は次のとおりです：

// MeastureStringとMeasureStringFMtは計算に役立ちます

// レイアウト目的のテキスト境界

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

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

//バッファへの基本テキストレンダリング

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

const wchar\_t \*text) const;

//バッファへの書式付きテキストレンダリング

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

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

//次のDrawStringの変形は色合いを提供する

//さまざまな色合いを指定できるパラメータ

//グレーの

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;

//グリフ固有のメソッドは単一グリフを正確に配置するために使用されます

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 ノート：

* シェードパラメータを持つ DrawString（） variantは、白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、メニューシステムで選択した行に白黒のテキストを使用する場合などです。DrawString（） は背景のピクセルを描画しないので、白いテキストを黒くするには、まず白い長方形を描画する必要があることを覚えておいてくださいMeasureString メソッドを使用して、必要な四角形の境界を決定する。
* MeasureGlyph（） そして DrawGlyph（） は個々のグリフ(絵として書かれた文字)を正確に配置するのに役立ちます。これらのメソッドはグリフの境界ボックスのみを使用され、隣接するグリフ間の間隔と通常のテキストフローのレイアウトに使用される垂直方向のオフセットは考慮しません。これにより、グリフを正確に配置することができます。たとえば、シンボル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している場合です。（FrontPanelDemoサンプルには MeasureGlyph（） とDrawGlyph（） ）の例がいくつかあります。
* RasterFontが提供する基本的なテキストのフローに満足していない場合は、基礎となるRasterGlyphSheet クラスを利用することを検討してください。このクラスは カスタムテキストフローの実装を記述するために使用できるForEachGlyph（）テンプレートを提供します。ForEachGlyph（） 使い方の最良の例 は、 さまざまなRasterFont::DrawString \*（） メソッドの実装にみることができます。

その他の実装上の注意：

* Xbox One X DevKitでは、::IsXboxFrontPanelAvailable（） はtrueを返し、::GetDefaultXboxFrontPanel は有効な IXboxFrontPanelControl オブジェクトを返します。もしそうでなければ、::IsXboxFrontPanelAvailable（） は falseを返し、 ::GetDefaultXboxFrontPanel は nullptrを返します（例：Xbox One、Xbox One S、または物理的なフロントパネルのない小売店のコンソールなど）
* すべてのフレームで前面パネルに表示する必要はありません（IXboxFrontPanelControl::PresentBuffer）代わりに、1つ以上のピクセルが変更されると表示する必要があります。したがって、サンプルには ディスプレイバッファに変更があるたびに設定されるm\_dirtyメンバが存在します
* また、変化があるときはいつでもライト状態を設定することだけが必要です。
* 前面パネルのバッファに直接アクセスすることはできません。代わりに、自身のバッファを管理し、自身のバッファのアドレスを IXboxFrontPanelControl::PresentBufferに渡す必要があります。Sample::CaptureFrontPanelScreen（）は 単純にDDSサーフェスのピクセルペイロードとしてm\_panelBufferの内容を使用しています。

# 更新履歴

2017 年 4 月、サンプルの初回リリース

# プライバシーステートメント

サンプルをコンパイルして実行すると、サンプルの実行可能ファイルの名前がMicrosoftに送信され、サンプルの使用状況の追跡に役立ちます。このデータ収集を無効にするには、Main.cppの「Sample Usage Telemetry」というラベルの付いたコードブロックを削除します。

マイクロソフトのプライバシーポリシー全般に関する詳細については、 [Microsoftのプライバシーステートメント](https://privacy.microsoft.com/ja-jp/privacystatement/)をご参照ください。