[UEFI Practice] UEFI Graphics Display (From Pixels to Characters)

原创 jiangwei0512 Posted on 2023-06-29 00:01:31 Read 2.1k Collection 5 Likes 1

Category Column: UEFI Development Basics Article Tags: uefi

摘要 This article describes how to package the pixel-level output of the graphics driver into character output in the UEFI environment, involving the implementation of EFI_DRIVER_BINDING_PROTOCOL, including the Supported and Start functions. The article explains in detail the process of converting from the graphics card's GOP protocol to text mode, including initializing text mode information, installing Simple TextOutProtocol, and showing how to convert Unicode strings to images on the screen.

GraphicsConsoleDxe

In [UEFI Practice] UEFI Graphics Display (Display Driver), we have introduced how to use the GOP installed by the graphics card driver to perform pixel-level display. This article introduces the packaging of pixels and eventually converts them into ordinary character output.

Module Description

This module packs the original GOP into a character output display module. The smallest unit of GOP output is pixel, and after packing, the smallest unit of text mode output becomes characters.

This module is also a UEFI Driver Model, corresponding to `EFI_DRIVER_BINDING_PROTOCOL` :

c AI generated projects 登录复制 run

```c
EFI_DRIVER_BINDING_PROTOCOL  gGraphicsConsoleDriverBinding = {
  GraphicsConsoleControllerDriverSupported,
  GraphicsConsoleControllerDriverStart,
  GraphicsConsoleControllerDriverStop,
  0xa,
  NULL,
  NULL
};
```

The supported function is a series of Protocol judgments, including:

- `gEfiGraphicsOutputProtocolGuid` Or `gEfiUgaDrawProtocolGuid` , as described in the previous section, `gEfiGraphicsOutputProtocolGuid` it will be installed and it will be used first.
- `gEfiDevicePathProtocolGuid` , for a PCI graphics card, this will also be installed.
- `gEfiHiiDatabaseProtocolGuid` Both `gEfiHiiFontProtocolGuid` are the basic interfaces of the UEFI user interface and are also required to use display output. In particular `gEfiHiiFontProtocolGuid` , it is the bridge between pixels and characters and is responsible for completing the conversion between the two.

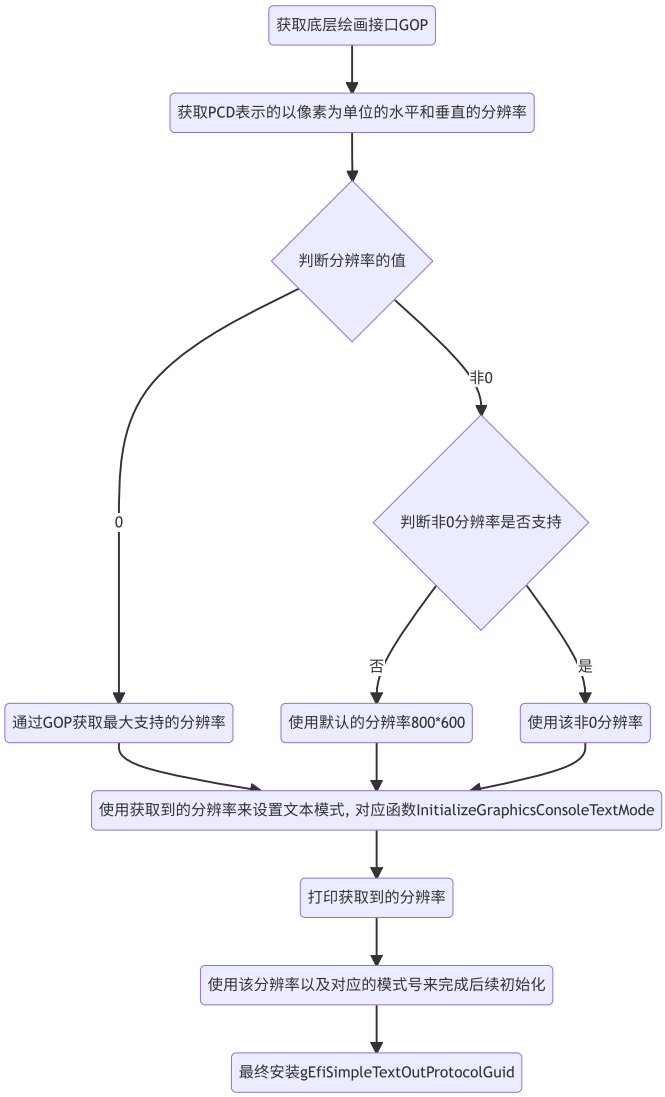
One of the main tasks of the Start function is to initialize the following structure:

c AI generated projects 登录复制 run

```c
//
// Graphics Console Device Private Data template
//
GRAPHICS_CONSOLE_DEV  mGraphicsConsoleDevTemplate = {
  GRAPHICS_CONSOLE_DEV_SIGNATURE,       // 一个标识，SIGNATURE_32 ('g', 's', 't', 'o')
  (EFI_GRAPHICS_OUTPUT_PROTOCOL *)NULL, // 显卡初始化模块安装的Protocol，实际绘制字体的接口
  (EFI_UGA_DRAW_PROTOCOL *)NULL,        // 如果上一个存在，这个就可以不需要了
  { // EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL基本接口，也是后续UEFI代码操作的接口，完成字符输出及相关操作
    GraphicsConsoleConOutReset,
    GraphicsConsoleConOutOutputString,
    GraphicsConsoleConOutTestString,
    GraphicsConsoleConOutQueryMode,
    GraphicsConsoleConOutSetMode,
    GraphicsConsoleConOutSetAttribute,
    GraphicsConsoleConOutClearScreen,
    GraphicsConsoleConOutSetCursorPosition,
    GraphicsConsoleConOutEnableCursor,
    (EFI_SIMPLE_TEXT_OUTPUT_MODE *)NULL // 它指向的就是下面的结构体
  },
  { // EFI_SIMPLE_TEXT_OUTPUT_MODE
    0,      // QueryMode()和SetMode()支持的模式数，由于没有初始化，所以现在默认是0
    -1,     // 当前的模式，-1表示的是无效的模式
    EFI_TEXT_ATTR (EFI_LIGHTGRAY,          EFI_BLACK),  // 当前字体输出属性，包括前景色和背景色
    0,      // 光标列位置
    0,      // 光标行位置
    FALSE   // 光标是否可见
  },
  (GRAPHICS_CONSOLE_MODE_DATA *)NULL,   // 这个看上去是一个指针，但是实际上是一个数组，表示当前文本显示支持的模式
  (EFI_GRAPHICS_OUTPUT_BLT_PIXEL *)NULL // 像素点的属性，其实是一个蓝绿红表示的值
};
```

收起 ^

The corresponding Start function process:



The whole process mainly consists of the following steps:

- Get the available resolutions;
- Initialize text mode information;
- Install the final Protocol.

Text Mode

Similar to GOP, text display also has its own mode (here mode represents GOP mode, and text mode represents SimpleTextOutProtocol mode), but it is much simpler than the former:

c AI generated projects 登录复制 run

```c
/**
  @par Data Structure Description:
  Mode Structure pointed to by Simple Text Out protocol.
**/
typedef struct {
  ///
  /// The number of modes supported by QueryMode () and SetMode ().
  ///
  INT32    MaxMode;

  //
  // current settings
  //

  ///
  /// The text mode of the output device(s).
  ///
  INT32      Mode;
  ///
  /// The current character output attribute.
  ///
  INT32      Attribute;
  ///
  /// The cursor's column.
  ///
  INT32      CursorColumn;
  ///
  /// The cursor's row.
  ///
  INT32      CursorRow;
  ///
  /// The cursor is currently visible or not.
  ///

```

```
  BOOLEAN    CursorVisible;
} EFI_SIMPLE_TEXT_OUTPUT_MODE;
```

收起 ^

The meaning of the specific parameters can be found in the English description or the previous comments. The display in GOP is in pixels, while the display in text mode is in small rectangles. Each rectangle contains a character. Here, the position is specified by Column and Row. The previous two structure members are similar `MaxMode` to `Mode` the modes in GOP. They also indicate that there are multiple and one of them is specified. This also means that there is another array to represent all supported text modes. They describe the relationship between pixels and character rectangles. The structure is represented as follows:

c AI generated projects 登录复制 run

```c
typedef struct {
  UINTN     Columns;        // 表示文本模式对应的列数
  UINTN     Rows;           // 表示文本模式对应的行数
  INTN      DeltaX;         // 文本显示相对于GOP显示的水平偏移
  INTN      DeltaY;         // 文本显示相对于GOP显示的垂直偏移
  UINT32    GopWidth;       // GOP显示的宽度，即水平像素个数
  UINT32    GopHeight;      // GOP显示的高度，即垂直像素个数
  UINT32    GopModeNumber;  // 对应GOP模式的Index
} GRAPHICS_CONSOLE_MODE_DATA;
```

This corresponds to the array mentioned above:

c AI generated projects 登录复制 run

```c
//
// Graphics Console Device Private Data template
//
GRAPHICS_CONSOLE_DEV  mGraphicsConsoleDevTemplate = {
  // 略
  (GRAPHICS_CONSOLE_MODE_DATA *)NULL,   // 这个看上去是一个指针，但是实际上是一个数组，表示当前文本显示支持的模式
  // 略
};
```

The above two structures constitute a complete text mode and are bound to the underlying GOP mode.

The following sample code shows all currently supported text modes:

c AI generated projects 登录复制 run

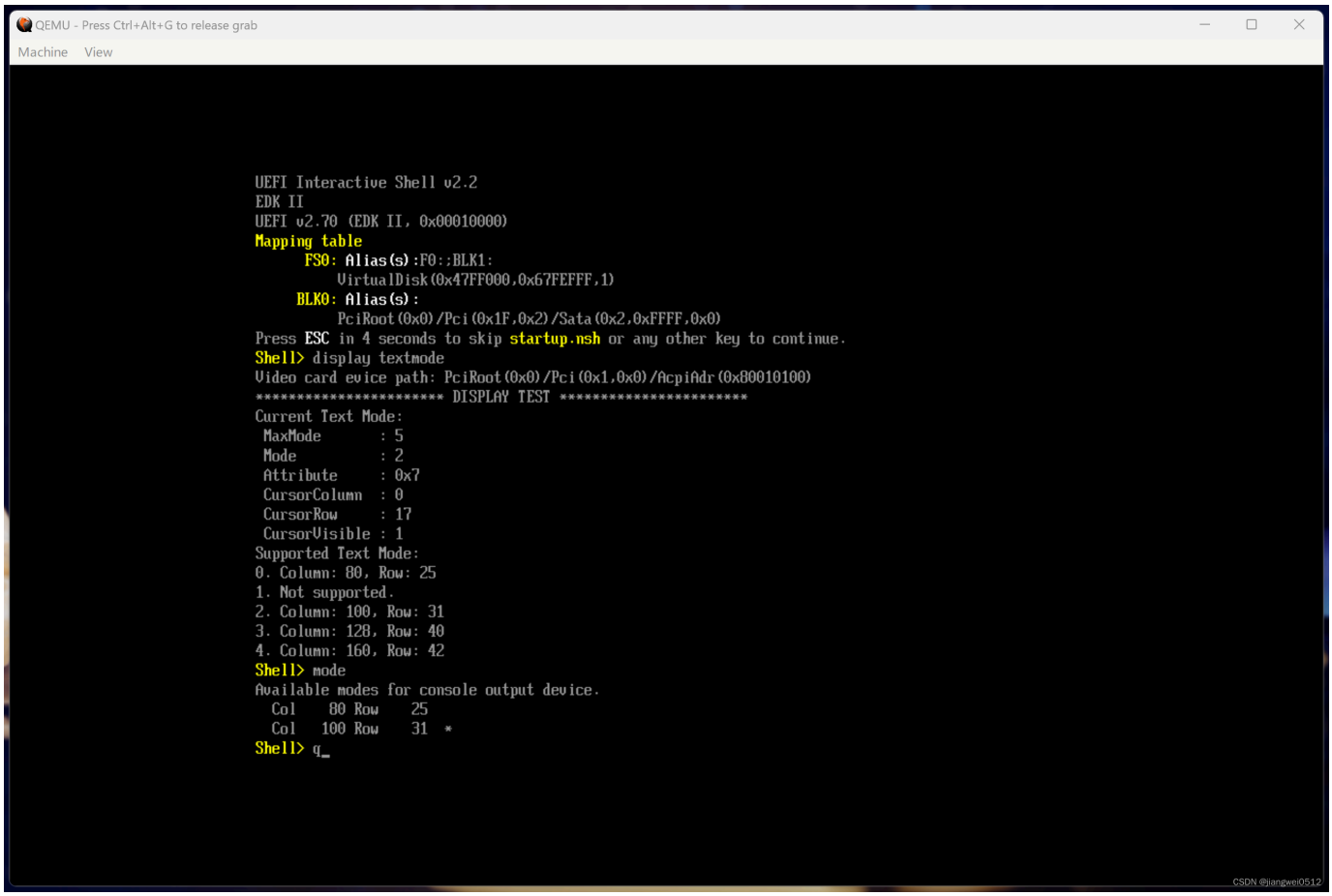
```c
VOID
ShowTextMode (
  IN  EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL *Stp
  )
{
  EFI_STATUS  Status = EFI_ABORTED;
  UINTN       Index = 0;
  UINTN       Col = 0;
  UINTN       Row = 0;

  Print (L"Current Text Mode:\r\n");
  Print (L" MaxMode       : %d\r\n", Stp->Mode->MaxMode);
  Print (L" Mode          : %d\r\n", Stp->Mode->Mode);
  Print (L" Attribute     : 0x%x\r\n", Stp->Mode->Attribute);
  Print (L" CursorColumn  : %d\r\n", Stp->Mode->CursorColumn);
  Print (L" CursorRow     : %d\r\n", Stp->Mode->CursorRow);
  Print (L" CursorVisible : %d\r\n", Stp->Mode->CursorVisible);

  Print (L"Supported Text Mode:\r\n");
  for (Index = 0; Index < Stp->Mode->MaxMode; Index++) {
    Status = Stp->QueryMode (Stp, Index, &Col, &Row);
    if (EFI_ERROR (Status)) {
      Print (L"%d. Not supported.\r\n", Index);
      continue;
    }
    Print (L"%d. Column: %d, Row: %d\r\n", Index, Col, Row);
  }
}
```

收起 ^

The result is:



There are a few points to note here:

1. Compared with the GOP `QueryMode()` function, the text mode `QueryMode()` can only view the row and column parameters, and the underlying ones `GRAPHICS_CONSOLE_MODE_DATA` cannot be viewed directly. If you want to obtain this information, you can view it by adding DEBUG information to this module:

```c
ModeData 0
Columns       : 80
Rows          : 25
DeltaX        : 320
DeltaY        : 162
GopWidth      : 1280
GopHeight     : 800
GopModeNumber : 0
ModeData 1
Columns       : 0
Rows          : 0
DeltaX        : 0
DeltaY        : 0
GopWidth      : 1280
GopHeight     : 800
GopModeNumber : 0
ModeData 2
Columns       : 100
Rows          : 31
DeltaX        : 240
DeltaY        : 105
GopWidth      : 1280
GopHeight     : 800
GopModeNumber : 0
ModeData 3
Columns       : 128
Rows          : 40
DeltaX        : 128
DeltaY        : 20
GopWidth      : 1280
GopHeight     : 800
GopModeNumber : 0
ModeData 4
Columns       : 160
Rows          : 42
DeltaX        : 0
DeltaY        : 1
GopWidth      : 1280
GopHeight     : 800
GopModeNumber : 0
```


2. Here you can see that a total of 5 are supported. As for why there are so many, it will be explained in the following chapters.
3. `mode` The command is to view the currently supported text modes. You can see that there are a few missing. The reason for this will be analyzed later.

Finally, let's explain this structure member in text mode `Attribute`. Its value can be divided into two categories. One is color, which corresponds to the pixel color of GOP; the other indicates whether the text is narrow or wide. An example of its corresponding is that English is narrow and Chinese is wide. Their difference leads to different pixels required to draw a font. The following section will introduce it further. `Attribute` Current value:

c AI generated projects 登录复制 run

```c
//
// EFI Console Colours
//
#define EFI_BLACK         0x00
#define EFI_BLUE          0x01
#define EFI_GREEN         0x02
#define EFI_CYAN          (EFI_BLUE | EFI_GREEN)
#define EFI_RED           0x04
#define EFI_MAGENTA       (EFI_BLUE | EFI_RED)
#define EFI_BROWN         (EFI_GREEN | EFI_RED)
#define EFI_LIGHTGRAY     (EFI_BLUE | EFI_GREEN | EFI_RED)
#define EFI_BRIGHT        0x08
#define EFI_DARKGRAY      (EFI_BLACK | EFI_BRIGHT)
#define EFI_LIGHTBLUE     (EFI_BLUE | EFI_BRIGHT)
#define EFI_LIGHTGREEN    (EFI_GREEN | EFI_BRIGHT)
#define EFI_LIGHTCYAN     (EFI_CYAN | EFI_BRIGHT)
#define EFI_LIGHTRED      (EFI_RED | EFI_BRIGHT)
#define EFI_LIGHTMAGENTA  (EFI_MAGENTA | EFI_BRIGHT)
#define EFI_YELLOW        (EFI_BROWN | EFI_BRIGHT)
#define EFI_WHITE         (EFI_BLUE | EFI_GREEN | EFI_RED | EFI_BRIGHT)
twen
twen//
twen// Macro to accept color values in their raw form to create
twen// a value that represents both a foreground and background
// color in a single byte.
// For Foreground, and EFI_* value is valid from EFI_BLACK(0x00) to
// EFI_WHITE (0x0F).
// For Background, only EFI_BLACK, EFI_BLUE, EFI_GREEN, EFI_CYAN,
// EFI_RED, EFI_MAGENTA, EFI_BROWN, and EFI_LIGHTGRAY are acceptable
//
// Do not use EFI_BACKGROUND_xxx values with this macro.
//
#define EFI_TEXT_ATTR(Foreground, Background)  ((Foreground) | ((Background) << 4))

#define EFI_BACKGROUND_BLACK      0x00
#define EFI_BACKGROUND_BLUE       0x10
#define EFI_BACKGROUND_GREEN      0x20
#define EFI_BACKGROUND_CYAN       (EFI_BACKGROUND_BLUE | EFI_BACKGROUND_GREEN)
#define EFI_BACKGROUND_RED        0x40
#define EFI_BACKGROUND_MAGENTA    (EFI_BACKGROUND_BLUE | EFI_BACKGROUND_RED)
#define EFI_BACKGROUND_BROWN      (EFI_BACKGROUND_GREEN | EFI_BACKGROUND_RED)
#define EFI_BACKGROUND_LIGHTGRAY  (EFI_BACKGROUND_BLUE | EFI_BACKGROUND_GREEN | EFI_BACKGROUND_RED)

//
// We currently define attributes from 0 - 7F for color manipulations
// To internally handle the local display characteristics for a particular character,
// Bit 7 signifies the local glyph representation for a character.  If turned on, glyphs will be
// pulled from the wide glyph database and will display locally as a wide character (16 X 19 versus 8 X 19)
// If bit 7 is off, the narrow glyph database will be used.  This does NOT affect information that is sent to
// non-local displays, such as serial or LAN consoles.
//
#define EFI_WIDE_ATTRIBUTE  0x80
```

收起 ^

The currently supported font types and colors are detailed here.

From GOP mode to text mode

From GOP mode to text mode, the first thing to solve is the problem of converting pixels into rows and columns mentioned in the previous chapter, which is mainly `InitializeGraphicsConsoleTextMode()` completed in the function:

c AI generated projects 登录复制 run

```c
EFI_STATUS
InitializeGraphicsConsoleTextMode (
  IN UINT32                     HorizontalResolution,
  IN UINT32                     VerticalResolution,
  IN UINT32                     GopModeNumber,
  OUT UINTN                     *TextModeCount,
  OUT GRAPHICS_CONSOLE_MODE_DATA  **TextModeData
  )
```

The input parameters of this function are the length and width of the pixel structure and the corresponding GOP mode, and the output parameter is the corresponding adaptable text mode. The following is a brief introduction to the implementation of this function to understand the conversion relationship from pixels to rows and columns.

1. First, determine the maximum supported columns and rows based on the number of horizontal and vertical pixels. The calculation relationship is as follows:

c AI generated projects 登录复制 run

```c
  MaxColumns = HorizontalResolution / EFI_GLYPH_WIDTH;  // 8
  MaxRows    = VerticalResolution / EFI_GLYPH_HEIGHT;   // 19
```

The 8 and 19 here come from the narrow font defined in the UEFI specification, which will be explained later. According to the requirements of the UEFI specification, the minimum supported rows and columns must meet the requirement of 80x25, so the following judgment is made:

c AI generated projects 登录复制 run

```c
  //
  // According to UEFI spec, all output devices support at least 80x25 text mode.
  //
  ASSERT ((MaxColumns >= 80) && (MaxRows >= 25));
```

Taking the current OVMF example, the pixel is 1280x800, so MaxColumns = 160, MaxRows = 42, it will be used as the rows and columns supporting full screen, so it will be placed `mGraphicsConsoleModeData` in, so all the rows and columns supported in the actual code are as follows:

```c
GRAPHICS_CONSOLE_MODE_DATA  mGraphicsConsoleModeData[] = {
  { 100, 31 },  //  800 x 600
  { 128, 40 },  // 1024 x 768
  { 160, 42 },  // 1280 x 800
  { 240, 56 },  // 1920 x 1080
  // 上面的都是硬编码的
  //
  // New modes can be added here.
  // The last entry is specific for full screen mode.
  //
  { 160,   42  }    // 代码根据像素实际生成的
};
```

收起 ^

However, these `mGraphicsConsoleModeData` are not the rows and columns that the final text mode can support (obviously there are duplications in the above table), so some processing is still needed here.

2. The first step is to add a few default rows and columns, mainly 80x25 and 80x50:

```c
  //
  // Mode 0 and mode 1 is for 80x25, 80x50 according to UEFI spec.
  //
  ValidCount = 0;

  NewModeBuffer[ValidCount].Columns       = 80;
  NewModeBuffer[ValidCount].Rows          = 25;
  NewModeBuffer[ValidCount].GopWidth      = HorizontalResolution;
  NewModeBuffer[ValidCount].GopHeight     = VerticalResolution;
  NewModeBuffer[ValidCount].GopModeNumber = GopModeNumber;
  NewModeBuffer[ValidCount].DeltaX        = (HorizontalResolution - (NewModeBuffer[ValidCount].Columns * EFI_GLYPH_WIDTH)) >> 1;
  NewModeBuffer[ValidCount].DeltaY        = (VerticalResolution - (NewModeBuffer[ValidCount].Rows * EFI_GLYPH_HEIGHT)) >> 1;
  ValidCount++;

  if ((MaxColumns >= 80) && (MaxRows >= 50)) {
    NewModeBuffer[ValidCount].Columns = 80;
    NewModeBuffer[ValidCount].Rows    = 50;
    NewModeBuffer[ValidCount].DeltaX  = (HorizontalResolution - (80 * EFI_GLYPH_WIDTH)) >> 1;
    NewModeBuffer[ValidCount].DeltaY  = (VerticalResolution - (50 * EFI_GLYPH_HEIGHT)) >> 1;
  }
twen
twen  NewModeBuffer[ValidCount].GopWidth      = HorizontalResolution;
twen  NewModeBuffer[ValidCount].GopHeight     = VerticalResolution;
twen  NewModeBuffer[ValidCount].GopModeNumber = GopModeNumber;
  ValidCount++;
```

收起 ^

Here we need to pay attention to the values of `DeltaX` and `DeltaY` , this is to ensure that even if the text mode cannot be full screen, it can still occupy the center of the screen.

We also need to pay attention to that `if` judgment. Obviously, this condition is met in the current OVMF environment, which results in that the rows and columns in the second item will not be assigned values, and are all defaulted to 0. This is quite strange, and the reason is not yet determined.

3. The following code starts to process `mGraphicsConsoleModeData` the text mode. The points to judge whether it is valid are: a) the number of rows and columns cannot exceed the maximum value, so `{ 240, 56 }` this item does not meet the requirement, b) there cannot be duplicate items. The main purpose here is to prevent the last item `{MaxColumns, MaxRows}` from conflicting with the previous one. The others are hard-coded in the code, and there should be no possibility of duplication between them.

The final available rows and columns are shown in the DEBUG information as follows:

```c
Graphics - Mode 0, Column = 80, Row = 25
Graphics - Mode 1, Column = 0, Row = 0
Graphics - Mode 2, Column = 100, Row = 31
Graphics - Mode 3, Column = 128, Row = 40
Graphics - Mode 4, Column = 160, Row = 42
```

This is also consistent with the previous analysis.

At this point, the pixels have been divided into rows and columns, and it can be seen that the most suitable situation is 160x42. However, from the previous examples and `mode` command printing, the actual use is 100x31. Since this does not affect the description of this section, we will not pay attention to it for the time being.

What needs to be paid attention to later is how to represent a character in a rectangle after the pixels are divided into rectangles (the narrow body corresponds to a rectangle of 8x19 pixels). This can be clearly seen in the following figure:

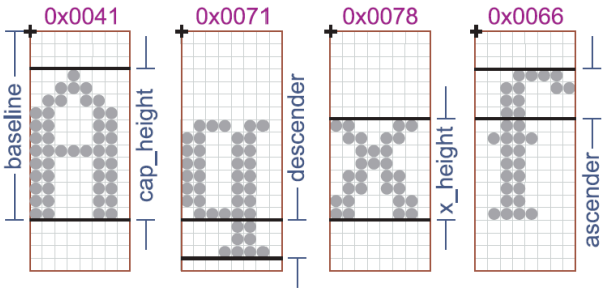


Figure 33-19 Font Description Terms

The smallest rectangle in the picture is a pixel, and the circle can be described by different colors (note that there is no real circle), so a word can be outlined. By observing the above picture, you can "write" a character through GOP. Here, take the character A as an example. You only need to change the pixels corresponding to the positions with circles in the above picture to other colors, and they can be displayed. Their corresponding positions are:

```c
UINT8                         BltIndex[NARROW_HEIGHT * NARROW_WIDTH] = {
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 1, 0, 0, 0, 0,
  0, 0, 1, 1, 1, 0, 0, 0,
  0, 1, 1, 0, 1, 1, 0, 0,
  1, 1, 0, 0, 0, 1, 1, 0,
  1, 1, 0, 0, 0, 1, 1, 0,
  1, 1, 0, 0, 0, 1, 1, 0,
  1, 1, 1, 1, 1, 1, 1, 0,
  1, 1, 0, 0, 0, 1, 1, 0,
  1, 1, 0, 0, 0, 1, 1, 0,
  1, 1, 0, 0, 0, 1, 1, 0,
  1, 1, 0, 0, 0, 1, 1, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0
twen};
```


收起 ^

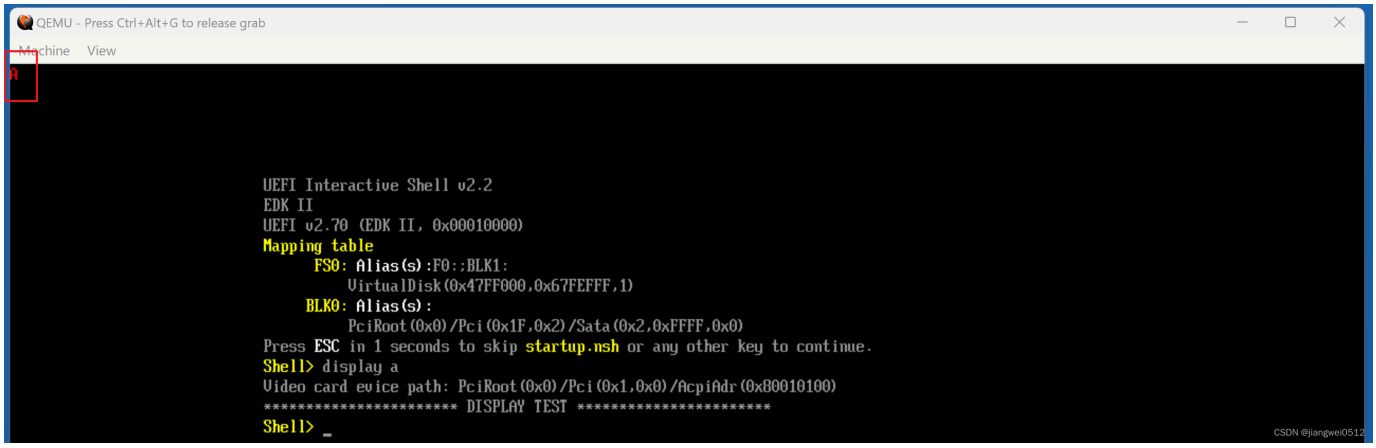
Here we use an array to simulate an 8x19 pixel. As long as the value is 1, it is represented by other colors. In this way, an A is constructed. Here is the remaining code:

```c
for (Index = 0; Index < NARROW_HEIGHT * NARROW_WIDTH; Index++) {
  if (BltIndex[Index]) {
    Blt[Index].Red = 0xFF;
  }
}

Gop->Blt (
      Gop,
      Blt,
      EfiBltBufferToVideo,
      0,
      0,
      0,
      0,
      Width,
      Height,
      0
      );
```


收起 ^

The final result is:



You can see an A is displayed in the upper left corner. However, this is just a simple example. How to output characters can be directly called by `EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL` the `OutputString()` function. Its implementation is:

c AI generated projects 登录复制 run

```c
EFI_STATUS
EFIAPI
GraphicsConsoleConOutOutputString (
  IN  EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL  *This,
  IN  CHAR16                           *WString
  )
```

Except for some special characters and special cases (for example, the Enter key means line break, and when the line break happens to be on the last line displayed, the whole system needs to be shifted upwards, and the parameter `EfiBltVideoToVideo` can be used at this time), the other processing is different in the following function:

c AI generated projects 登录复制 run

```c
/**
  Draw Unicode string on the Graphics Console device's screen.

  @param  This                  Protocol instance pointer.
  @param  UnicodeWeight         One Unicode string to be displayed.
  @param  Count                 The count of Unicode string.

  @retval EFI_OUT_OF_RESOURCES  If no memory resource to use.
  @retval EFI_UNSUPPORTED       If no Graphics Output protocol and UGA Draw
                                protocol exist.
  @retval EFI_SUCCESS           Drawing Unicode string implemented successfully.

**/
EFI_STATUS
DrawUnicodeWeightAtCursorN (
  IN  EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL  *This,
  IN  CHAR16                           *UnicodeWeight,
  IN  UINTN                            Count
  )
```

收起 ^

Converting characters into pixels depends on `EFI_HII_FONT_PROTOCOL`:

c AI generated projects 登录复制 run

```c
   Status = mHiiFont->StringToImage (
                        mHiiFont,
                        EFI_HII_IGNORE_IF_NO_GLYPH | EFI_HII_DIRECT_TO_SCREEN | EFI_HII_IGNORE_LINE_BREAK,
                        String,
                        FontInfo,
                        &Blt,
                        This->Mode->CursorColumn * EFI_GLYPH_WIDTH + Private->ModeData[This->Mode->Mode].DeltaX,
                        This->Mode->CursorRow * EFI_GLYPH_HEIGHT + Private->ModeData[This->Mode->Mode].DeltaY,
                        NULL,
                        NULL,
                        NULL
                        );
```

收起 ^

This function not only performs conversion, but also completes the final output. The implementation of HII Font will be further introduced later.