UEFI Development Exploration 96 – Thermometer Game

原创 luobing4365 Posted on 2021-08-20 22:18:53 Read 947 collect Likes 1

Category columns: UEFI Development Article Tags: uefi bios Low-level application development UEFI Programming Practice

摘要 This article introduces how to port and write a thermometer game in the UEFI environment, including image loading, display, and user operation processing. Use the LoadBitmapFile() function to load image resources, and use CopyBitmapToBitmap() to process image display. Users can control the rise and fall of the mercury column by pressing the up and down keys. The article provides complete code examples and compilation test methods.

The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

Thermometer game under UEFI

There are several small programs in syslibforuefi. From my point of view, the most worthwhile thing to learn is how to turn images into HII resources. I have already studied a lot of other graphics and images in my blog a long time ago.

When I opened CSDN this morning, I was surprised:

Figure 1 CSDN's evaluation

It has quietly climbed to the second place on the gaming content list!

I said last time that this is a blog about UEFI development, not game development. This weird comment made me not know where to start complaining.

Would it be better for me to switch to game development?

Forget it, today I will port the thermometer game in syslibforuefi to see the effect. This is the last article of exploring syslibforuefi, and I have read almost all the code.

1 Programming

The thermometer program is an ImageStack project of syslibforuefi. Its main functions are as follows:
1) Display the thermometer image;
2) When the user presses the up or down key, the mercury part of the thermometer rises or falls accordingly.

In other words, the part that needs to be dynamically processed is only the mercury part, and the rest can be treated as background patterns.

The function used to achieve image stretching and shrinking is StrethcImage(), and the display function is DisplayImageStack(). DisplayImageStack() is different from the functions of the same name in the previous two articles. Its implementation has been modified. The specific implementation can be viewed in my newly created project (see the address at the end of the article).

2 Porting and writing code

The original code still uses HII resources, which needs to be modified. To facilitate porting, all codes are in one source file, not split according to function. The basic porting steps are as follows:

1) Re-implement the image loading method

There are three images that need to be loaded: bg.bmp, front.bmp and mid.bmp. The goal of loading is to copy the image to the memory pointed to by the EFI_GRAPHICS_OUTPUT_BLT_PIXEL pointer. Therefore, you can use LoadFile() and LoadBitmapFile() from the previous article.

Change the SetUpImages() in the original project to the following:

```c
EFI_STATUS
SetUpImages (
```

```
  IN CONST CHAR8    *BgLogoFilePath,
  IN CONST CHAR8    *MidLogoFilePath,
  IN CONST CHAR8    *FrontLogoFilePath)
{
  // Initialize the graphics support.
  if (!InitGop()) {
    return EFI_UNSUPPORTED;
  }

  // Load the background bitmap.
  if (!LoadBitmapFile(BgLogoFilePath, &mBgBlt, &mBgHeight, &mBgWidth)) {
    return EFI_UNSUPPORTED;
  }


  if (!LoadBitmapFile(MidLogoFilePath, &mMidBlt, &mMidHeight, &mMidWidth)) {
    return EFI_UNSUPPORTED;
  }
  MaxHeight = mMidHeight;

  if (!LoadBitmapFile(FrontLogoFilePath, &mFrontBlt, &mFrontHeight, &mFrontWidth)) {
    return EFI_UNSUPPORTED;
  }

  return EFI_SUCCESS;
}
```

收起 ^

2) Image display function

Since the image will change, a processing function CopyBitmapToBitmap() is added to handle the changes of the mercury part.

c AI generated projects 登录复制 run

```
EFI_STATUS CopyBitmapToBitmap (
             IN OUT EFI_GRAPHICS_OUTPUT_BLT_PIXEL *Dest,
             IN UINTN DestWidth,
             IN UINTN DestHeight,
             IN EFI_GRAPHICS_OUTPUT_BLT_PIXEL *Src,
             IN UINTN SrcWidth,
             IN UINTN SrcHeight,
             IN UINTN OffsetX,
             IN UINTN OffsetY
             )
{
  UINTN CurrentX;

```

```
  UINTN CurrentY;
  UINTN SourceX;
  UINTN SourceY;

  EFI_GRAPHICS_OUTPUT_BLT_PIXEL Current;
  Current.Red = 0;
  Current.Green = 0;
  Current.Blue = 0;

  SourceX = 0;
  SourceY = 0;
  CurrentX = OffsetX;
  CurrentY = OffsetY;

  if (CurrentX > DestWidth || CurrentY > DestHeight) {
    return EFI_INVALID_PARAMETER;
  }

  //copy image
  //if source would go over edge of destination buffer, then clip edges

  while (SourceY < SrcHeight && CurrentY < DestHeight) {

    while (SourceX < SrcWidth && CurrentX < DestWidth) {
      Current = Src[(SourceY * SrcWidth) + SourceX];

      if (! (Current.Red == 0 && Current.Green == 0 && Current.Blue == 0)) {
        Dest[(CurrentY * DestWidth) + CurrentX] = Src[(SourceY * SrcWidth) + SourceX];
      }
        SourceX++;
        CurrentX++;
    }
    SourceX = 0;
    CurrentX = OffsetX;

    SourceY++;
    CurrentY++;

  }

  return EFI_SUCCESS;

}
```

收起 ^

The image display function DisplayImageStack() calls CopyBitmapToBitmap() to perform actual display processing.

```c
EFI_STATUS DisplayImageStack ()
{
  UINTN CoordinateX;
  UINTN CoordinateY;

  EFI_GRAPHICS_OUTPUT_BLT_PIXEL *Blt;

  Blt = (EFI_GRAPHICS_OUTPUT_BLT_PIXEL *) malloc(mBgWidth * mBgHeight * sizeof(EFI_GRAPHICS_OUTPUT_BLT_PIXEL));
  memset (Blt, 0, mBgWidth * mBgHeight * sizeof(EFI_GRAPHICS_OUTPUT_BLT_PIXEL));

  CopyBitmapToBitmap (Blt, mBgWidth, mBgHeight, mBgBlt, mBgWidth, mBgHeight, 0, 0);

  CoordinateX = (mBgWidth - mMidWidth) / 2;
  CoordinateY = 15 + (MaxHeight - mMidHeight);
  CopyBitmapToBitmap (Blt, mBgWidth, mBgHeight, mMidBlt, mMidWidth, mMidHeight, CoordinateX, CoordinateY);

  CoordinateX = (mBgWidth - mFrontWidth) / 2;
  CoordinateY = 15;
  CopyBitmapToBitmap (Blt, mBgWidth, mBgHeight, mFrontBlt, mFrontWidth, mFrontHeight, CoordinateX, CoordinateY);

  CoordinateX = (mGraphicsOutput->Mode->Info->HorizontalResolution / 2) - (mBgWidth / 2);
  CoordinateY = (mGraphicsOutput->Mode->Info->VerticalResolution / 2) - (mBgHeight / 2);

  mGraphicsOutput->Blt (
                    mGraphicsOutput,
                    Blt,
                    EfiBltBufferToVideo,
                    0,
                    0,
                    CoordinateX,
                    CoordinateY,
                    mBgWidth,
                    mBgHeight,
                    mBgWidth * sizeof (EFI_GRAPHICS_OUTPUT_BLT_PIXEL)
                    );

  return EFI_SUCCESS;

}
```


收起 ^

3) User operation processing

The main function main() calls StretchImage() to process the up and down arrow keys input by the user to control the rise and fall of mercury. The implementation is as follows:


```c
EFI_STATUS StretchImage ()
{
  EFI_INPUT_KEY Key;
  UINTN index;

  //clear screen
  gST->ConOut->Reset (gST->ConOut, FALSE);

  DisplayImageStack();

  //input loop
  do {
    gBS->WaitForEvent (1, &gST->ConIn->WaitForKey, &index);
    gST->ConIn->ReadKeyStroke (gST->ConIn, &Key);

    if (Key.ScanCode != SCAN_NULL) {
      if (Key.ScanCode == SCAN_UP) {
        // Print(L"UP");
        RotateUp ();
      } else if (Key.ScanCode == SCAN_DOWN) {
        // Print(L"Down");
        RotateDown();
      }
    }

    DisplayImageStack();

  } while (Key.ScanCode != SCAN_END);

  //clear screen after app exits
  gST->ConOut->Reset (gST->ConOut, FALSE);

  return EFI_SUCCESS;
}
```


The processing functions for the mercury's descent and rise are RotateDown() and RotateUp(), which change the length of the mercury part and display it through DisplayImageStack().

3 Test Thermometer Game

Compile using the following command:

```
C:\vUDK2018\edk2>build -p RobinPkg\RobinPkg.dsc -m RobinPkg\Applications\ImageStack\ImageStack.inf -a IA32
```

Copy the compiled program ImageStack.efi and bg.bmp, front.bmp and mid.bmp in the project folder to the folder where the simulator is located, and run ImageStack.efi to see the effect.

In the Tianocore simulator, the running effect is shown in Figure 2.

Figure 2 Thermometer game

After writing it, I feel that the function is a bit simple, and it is very similar to a program of GuiLite that I have ported before. If you use the graphics library written before in the blog, it can also be easily built.

I will not explore the syslibforuefi code any further. Interested netizens can download and read it by themselves.

Gitee address: https://gitee.com/luobing4365/uefi-explorer
Project code is located in: /FF RobinPkg/RobinPkg/Applications/ImageStack