

SET2 SET4/SET5 への プログラムの移行に関して

1998/6/30 セガ・ソフトウェア技術開発部

更新履歴 1998/6/30

pdExecPeripheralServer();は njWaitVSync();に入りました。

マクロのネストは対応

テクスチャバッファに関しての記述追加

更新履歴 1998/5/7

Set2NINJA と Set4NINJA の動作の違いに、マトリクス構造体の中身の変更に関する記述を追加

N I N J A に依存する部分の変更

インクルードファイルの変更

【変更内容】

"NINJA.H"を"SHINOBI.H"に変更する必要があります。

"SHINOBI.H"内で"NINJA.H"を呼び出しています。

初期化の変更

【変更内容】

「njExitSystem」を「sbExitSystem」に変更する必要があります。

引数については、まったく変更する必要はありません。

それと同時に

「sbinit.o」をリンクする必要があります。

【注意】

ファイルシステムを使用しファイルの読み込みを行う必要がある場合、「sbinit.o」を、
ファイルシステムを利用しない場合は「sbinitn.o」をリンクしてください。

スタート部分

【変更内容】

スタートアップモジュール

「[strt1.obj](#)」「[strt2.obj](#)」

をリンクする必要があります。

(サンプルに付属する makefile を参照してください)

エントリ関数の変更

SET2 環境ではエントリ関数は、「WinMain」でしたが、これを「main」関数に変更し。

以下のような追加が必要です。

```
void main(void)
{
    njUserInit();
    while (1) {
        if (njUserMain() < 0) break;
        njWaitVSync();
    }
    njUserExit();
}
```

サンプルで使用している「njloop.c」を使うと便利です。

【注意】

これは S e t 2 の N I N J A 環境では W I N 3 2 環境なので「WinMain」が必要でしたが、S e t 4 の SHINOBI 環境は S H C 環境に変わり、「main」を呼ぶようになった為です。

アライメント

【変更内容】

Ninja で使用する、

マトリックスのバッファを 8 B Y T E アライメント

バーテックスバッファを 3 2 B Y T E アライメント

に変更する必要があります。

【変更方法】

サンプルで使用している「global32.src」を使用する必要があります。

そしてプログラムの記述を

```
NJS_MATRIX    matrix[128];
NJS_VERTEX_BUF    vbuf[4096];
```

から

```
extern NJS_MATRIX    matrix[128];
```

```
extern NJS_VERTEX_BUF vbuf[4096];
```

に変更が必要です。

global32.src でアラインメントを合わせてマトリクスバッファとバーテックスバッファを確保し、それを外部宣言の形で C のソースに取り込みます。

テクスチャ

【変更内容】

Set 4 からは、テクスチャのバッファを宣言する必要があります。

【変更方法】

```
njInitTextureBuffer((Sint8 *)vbuf,0x40800);
```

【バッファのサイズに関して】

バッファにサイズに関しては

Katana/Shinobi/doc/02graph/05textur/01textur.doc"テクスチャガイドの

4.3 テクスチャバッファの設定」を参照して下さい。

(注意)デフォルトで「global32.src」の中を見て解るように vbuf は 4096*96Byte

しか宣言されておりません、バッファサイズを 4096*96Byte 以上に変

更する場合は注意して下さい、変更する部分は「global32.src」の

>_vbuf:

> .RES.B (4096*96)

とユーザーアプリの extern 宣言している部分です。

```
>extern NJS_VERTEX_BUF vbuf[4096];
```

関数の制限

【変更内容】

「sprintf」は使用できません。

Set2NINJA と Set4NINJA の動作の違い

光源計算

【変更内容】

光源の計算方法が変更になりました。

【その他】

スケールの変更がない場合は見た目は変わりません。

NJA (NINJAグラフィックフォーマット)

【変更内容】

バージョンが0.7.2以前のNINJAファイルには対応していません。

マトリックス構造体

【変更内容】

マトリックスがSet4からSH7091用にカスタマイズしたために、マトリックスの構造体の中身が変化しました。(以下の通りです)

(Set2)

```
enum {M 00,M 10,M 20,M 30,  
      M 01,M 11,M 21,M 31,  
      M 02,M 12,M 22,M 32,  
      M 03,M 13,M 23,M 33,  
      V 00,V 10,V 20,V 30,  
      V 01,V 11,V 21,V 31,  
      V 02,V 12,V 22,V 32,  
      V 03,V 13,V 23,V 33};
```

(Set4)

```
enum {M 00,M 01,M 02,M 03,  
      M 10,M 11,M 12,M 13,  
      M 20,M 21,M 22,M 23,  
      M 30,M 31,M 32,M 33};
```

NINJAの標準のライブラリーだけの開発であれば問題はないのですが個々でマトリックスを計算している場合に関しては、問題になります。

ファイルシステムの変更

ファイルシステム

【変更内容】

ファイルシステムを使用している場合は

gdFsOpen	ファイルのオープン
gdFsClose	ファイルのクローズ
gdFsRead	ファイルのリード
gdFsGetFileSize	ファイルサイズの取得

を使用して置き換えてください。

コンパイラに依存する部分の変更

コメント

//には対応をしていません。/**/をつかってください。

キャスト

キャストが正確に設定されていない場合エラーになります。

ストラクチャ

Win32 環境と SHC 環境では、構造体のメンバのデータアラインメントが異なります。
Intel CPU では、奇数アドレスからのワードまたはロングワードアクセスを許可していますが、SH では認められません。従って、構造体のメンバのデータサイズによってはメンバのバイトアラインメントが Win32 のものとは違ってきます。

もし、構造体のメンバの位置や構造体サイズを利用しているアプリケーションの場合、十分に注意する必要があります。

SH 環境の場合アドレスエラーが発生する具体例

(例)

```
char    a[10];  
short   *b      =&a[1];  
*b      =0;      <-この部分でアドレスエラーが発生します。
```

Define マクロの精度(浮動小数)

(例)

```
#define PI      =3.14159265358982.....
```

というような高い精度の代入には対応していません。

(対応)

```
#define PI      =3.141592
```

精度を下げてください。

メモリ管理

SHINOBI ライブラリでは、メモリアロケーション関数「syMalloc」を用意しています。

メモリアロケーションについてはこの関数を使ってください。

SyMalloc 関数は sbInitSystem でシステムを初期化した後につかいます。

アライメント

S H Cはアライメントに関する対応が容易ではありません。

以下の3つの方法でアライメントに対応したメモリの確保が出来ます。

1 . アセンブラによる対応

```
.ALIGN      32
```

を使用してアライメントの定義が実現します。

サンプルで使用している「global32.src」中の記述が参考になります。

2 . Cの記述による対応

(例)

```
Uint8 temp_data[data_size + 31];
```

```
temp  = (Uint8*)((((Uint32)temp_data & 0xfffffe0) + 0x20);
```

という記述で対応は可能です。

3 . syMalloc による対応

syMalloc を使用する事で、アライメントは保証されます。

以上