Wire-Frame 3D Graphics Accelerator IP Core

C Library Specification

Kenji Ishimaru <kenji.ishimaru@prtissimo.com>

Revision History

Rev.	Date	Author	Description
1.0	2015/09/30	Kenji Ishimaru	First release

Contents

1 Introduction

This document describes C Application Programming Interface (API) for Wire-Fame 3D graphics Accelerator IP Core. The API makes it easy to generate Model-View and Projection matrices, and makes it easy to handle 3D rendering process.

1 - 1 File List

1 - 1 - 1 Common Files

File Name	Directory	Description
mp_lib.h	clib/hw_dep	Graphics API implementation
mp_lib.c		
mp_matrix3.h	clib	3x3 matrix utility
mp_matrix3.c		
mp_matrix4.h		4x4 matrix utility
mp_matrix4.c		
mp_vector3.h		3-element vector utility
mp_vector3.c		
mp_vector4.h		4-element vector utility
mp_vector4.c		

1 - 1 - 2 Hardware Dependent Files

File Name	Description
mp_hwdep.h	hardware-dependent function
mp_hwdep.c	

2 Parameters

2 - 1 Matrix Stack Size

define name	file	default value
MP_CONFIG_MAX_MSTACK	mp_lib.c	8

MP_CONFIG_MAX_MSTACK specifies the size of the matrix stack. mpPushMatrix pushes a matrix to the stack, and mpPopMatrix pops a matrix form the stack. Note that Model-View matrix and Projection matrix shares same matrix stack.

2 - 2 Buffer Address

define name	file	Description
FRAME_BUFFER_0	mp_hwdep.h	Frame Buffer0 top Address
FRAME_BUFFER_1		Frame Buffer1 top Address

3D graphics rendering typically uses double frame buffers (front and back buffers). The one is assigned to 3D graphics rendering, and the other is assigned to LCD controller to displaying previous rendering result. Each buffer should have enough space to allocate maximum screen size. For example, if the screen size is VGA(640x480), Each buffer has space at least 0x4b000 byte.

2 - 3 Hardware Dependent

The top address of the IP Core is system dependent. All the register addresses in the IP Core is defined by offset from the top address. For further details about each register, please refer “Wire-Frame 3D Graphics Accelerator IP Core Specification”.

define name	offset from register address top
D3D_DMA_START	0x00
D3D_DMA_STATUS	0x04
D3D_DMA_ADRS	0x08
D3D_DMA_SIZE	0x0c
D3D_MATRIX_M00 –	0x10 - 0x4c

D3D_MATRIX_M33	
D3D_FSCR_W	0x50
D3D_FSCR_H	0x54
D3D_ISCR_W_M1	0x58
D3D_ISCR_H_M1	0x5c
D3D_ISCR_W	0x60
D3D_COL_ADRS	0x64
D3D_COL_VAL	0x68

3 API List

3 - 1 3D Graphics Core API

Name	Description
mpMatrixMode	specify current matrix mode
mpPushMatrix	push the current matrix stack
mpPopMatrix	pop the current matrix stack
mpMultMatrix	multiply the current matrix by an arbitrary matrix
mpLoadIdentity	set the current matrix with the identity matrix
mpTranslate	multiply the current matrix by a translation matrix
mpRotate	multiply the current matrix by a rotation matrix
mpScale	multiply the current matrix by a general scaling matrix
mpViewPort	set the viewport
mpFrustum	multiply a perspective matrix
mpOrtho	multiply an orthographic matrix
mpPerspectice	set up a perspective projection matrix
mpLookAt	define a viewing matrix
mpVertexPointer	define an array of vertex data
mpDrawArrays	render triangles from array data
mpRenderColor	set the current render color
mpRenderColorU	set the current render color by unsigned byte

3 - 2 Helper API

Name	Description
mpClearColor	specify clear values for the frame buffer
mpClear	clear frame buffer (back buffer)
mpSwapBuffers	exchange front and back buffers

3-3 mpMatrixMode

3-3-1 C Specification

void mpMatrixMode(enum mode)

3-3-2 Parameters

Parameter	Description
mode	current matrix for subsequent matrix operations. Two values are accepted: MP_MODELVIEW and MP_PROJECTION.

3-3-3 Description

mode can assume one of two values:

MP_MODELVIEW

Applies subsequent matrix operations to the modelview matrix.

MP_PROJECTION

Applies subsequent matrix operations to the projection matrix.

3-3-4 Notes

None

3-3-5 Examples

```
mpMatrixMode(MP_MODELVIEW);
mpMatrixMode(MP_PROJECTION);
```

3-4 mpPushMatrix

3-4-1 C Specification

void mpPushMatrix(void)

3-4-2 Parameters

None

3-4-3 Description

mpPushMatrix pushes the current matrix to the matrix stack.
The only one current matrix exists. This means that mpMatrixMode call between mpPushMatrix and mpPopMatrix is unsupported.

3-4-4 Notes

mpPushMatrix to the full stack is ignored.

3-4-5 Examples

```
mpPushMatrix()
        : (some matrix operation, mpTranslate(), mpRotate(), etc.)
mpPopMatrix()
```

3-5 mpPopMatrix

3-5-1 C Specification

void mpPopMatrix(void)

3-5-2 Parameters

None

3-5-3 Description

The only one current matrix exists. This means that mpMatrixMode call between mpPushMatrix and mpPopMatrix is unsupported.

3-5-4 Notes

The behavior of empty stack pop is undefined.

3-5-5 Examples

```
mpPushMatrix()
        : (some matrix operation, mpTranslate, mpRotate, etc.)
mpPopMatrix()
```

3-6 mpLoadIdentity

3-6-1 C Specification

void mpLoadIdentity(void)

3-6-2 Parameters

None

3-6-3 Description

mpLoadIdentity replaces the current matrix with the identity matrix.

$$current\ matrix = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3-6-4 Notes

None

3-6-5 Examples

```
mpLoadIdentity();
```

3-7 mpTranslate

3-7-1 C Specification

void mpTranslate(float x, float y, float z);

3-7-2 Parameters

parameter	Description
x,y,z	x, y, and z coordinates of a translation vector.

3-7-3 Description

mpTranslate translates current matrix by x y z.

The operation is:

$$\text{CurrentMatrix} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix}$$

$$\begin{pmatrix} tx \\ ty \\ zt \\ tw \end{pmatrix} = \text{CurrentMatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\text{CurrentMatrix} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & \text{tx} \\ m_{10} & m_{11} & m_{12} & \text{ty} \\ m_{20} & m_{21} & m_{22} & \text{tz} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix}$$

3-7-4 Notes

None

3-7-5 Examples

mpTranslate(1.0, 2.0, 0.0);

3-8 mpRotate

3-8-1 C Specification

void mpRotate(float angle, float x, float y, float z);

3-8-2 Parameters

Parameter	Description
angle	angle of rotation, in degrees.
x,y,z	x, y, and z coordinates of a vector.

3-8-3 Description

mpRotate produces a rotation of angle degrees around the vector x,y,z.
The operation is:

$\theta = \text{angle}/\pi$

$RotateMatrix$

$$= \begin{pmatrix} x^2(1-\cos\theta)+\cos\theta & \text{xy}(1-\cos\theta)-\text{zsin}\theta & \text{xz}(1-\cos\theta)+\text{ys} & 0 \\ \text{yx}(1-\cos\theta)+\text{zsin}\theta & y^2(1-cos\theta)+cos\theta & \text{yz}(1-\cos\theta)-\text{xsin}\theta & 0 \\ \text{xz}(1-\cos\theta)-\text{ysin}\theta & \text{yz}(1-\cos\theta)+\text{xsin}\theta & z^2(1-cos\theta)+cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\text{CurrentMatrix} = \text{CurrentMatrix} \cdot \text{RorateMatrix}$

3-8-4 Notes

None

3-8-5 Examples

```
mpRotate(90, 0, 1, 0);        // rotate 90 degree around y axis
```

3-9 mpScale

3-9-1 C Specification

void mpScale(float x, float y, float z);

3-9-2 Parameters

Parameter	Description
x,y,z	scale factors along the x, y, and z axes

3-9-3 Description

mpScale produces a nonuniform scaling along the x, y, and z axes.
The operation is:

$$\text{CurrentMatrix} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix}$$

$$\text{CurrentMatrix} = \text{CurrentMatrix} \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3-9-4 Notes

None

3-9-5 Examples

mpScale(2.0,2.0,2.0);

3-10 mpViewPort

3-10-1 C Specification

void mpViewPort(int width, int height)

3-10-2 Parameters

Parameter	Description
width, height	Specify the width and height of the viewport.

3-10-3 Description

mpViewport specifies the transformation of x and y from normalized device coordinates to window coordinates.
The operation is:

$$x_w = (x_{nd} + 1)\frac{width}{2}$$

$$y_w = (y_{nd} + 1)\frac{heigh}{2}$$

Where:

x_{nd}, y_{nd}: $normalized\ device\ coordinates$
x_w, y_w: $window\ coordinates$

3-10-4 Note

Viewport transformation is calculated in the IP Core.

3-10-5 Examples

mpViewPort(640,480); // VGA size

3-11 mpFrustum

3-11-1 C Specification

void mpFrustum(float left, float right, float bottom, float top, float nearVal, float farVal);

3-11-2 Parameters

Parameter	Description
left, right	The coordinates for the left and right vertical clipping planes.
bottom, top	The coordinates for the bottom and top horizontal clipping planes.
nearVal, farVal	The distances to the near and far depth clipping planes.

3-11-3 Description

mpFrustum produces a perspective projection matrix.
The operation is:

$$\text{ProjMatrix} = \begin{pmatrix} \frac{2nearVal}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2nearVal}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{farVal+nearVal}{farVal-nearVal} & -\frac{2farValnearVal}{farVal-nearVal} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$\text{CurrentMatrix} = \text{CurrentMatrix} \cdot \text{ProjMatrix}$$

3-11-4 Notes

None

3-11-5 Examples

mpFrustum is called in mpPerspective

```
mpPerspective(30.0, 4.0 / 3.0, 1, 100);
```

3-12 mpOrtho

3-12-1 C Specification

void mpOrtho(float left, float right, float bottom, float top, float nearVal, float farVal);

3-12-2 Parameters

Parameter	Description
left, right	The coordinates for the left and right vertical clipping planes.
bottom, top	The coordinates for the bottom and top horizontal clipping planes.
nearVal, farVal	The distances to the near and far depth clipping planes.

3-12-3 Description

mpOrtho produces a parallel projection matrix.
The operation is:

$$\text{ProjMatrix} = \begin{pmatrix} \frac{2}{right-left} & 0 & 0 & \frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & \frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{farVal-nearVal} & -\frac{nearVal+farVal}{farVal-nearVal} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{CurrentMatrix} = \text{CurrentMatrix} \cdot \text{ProjMatrix}$$

3-12-4 Notes

None

3-12-5 Examples

3-13 mpPerspective

3-13-1 C Specification

void mpPerspective(float fovy, float aspect, float zNear, float zFar)

3-13-2 Parameters

Parameter	Description
fovy	The field of view angle in degrees,in the y direction.
aspect	The aspect ratio that determines the field of view in the x direction.
zNear	The distance from the viewer to the near clipping plane
zFar	The distance from the viewer to the far clipping plane

3-13-3 Description

mpPerspective specifies a viewing frustum into the world coordinate system. The aspect ratio in mpPerspective should match the aspect ratio of mpViewport.

The operation is:

$$\text{ymax} = \text{zNear} \times \tan(\frac{fovy \times \pi}{360})$$

$$\text{xmax} = \text{ymax} \times \text{aspect}$$

mpFrustumf(-xmax, xmax, -ymax, ymax, zNear, zFar);

3-13-4 Notes

None

3-13-5 Examples

mpPerspective(30.0, 4.0 / 3.0, 1, 100);

3-14 mpLookAt

3-14-1 C Specification

void mpLookAt(float eyeX, float eyeY, float eyeZ, float centerX, float centerY, float centerZ, float upX, float upY, float upZ);

3-14-2 Parameters

Parameter	Description
eyeX, eyeY, eyeZ	The position of the eye point
centerX, centerY, centerZ	The position of the reference point
upX, upY, upZ	The direction of the up vector

3-14-3 Description

mpLookAt creates a viewing matrix derived from an eye point, a reference point indicating the center of the scene, and an UP vector.
The operation is:

$$M = \begin{pmatrix} s.x & u.x & -f.x & 0 \\ s.y & u.y & -f.y & 0 \\ s.z & u.z & -f.z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\mathrm{CurrentMatrix} = \mathrm{CurrentMatrix} \cdot \mathrm{M}$
mpTranslate(-eyeX,-eyeY,-eyeZ)

Where:

$\mathrm{f.x} = (\mathrm{centerX} - \mathrm{eyeX}), \mathrm{f.y} = (\mathrm{centerY} - \mathrm{eyeY}), \mathrm{f.z} = (\mathrm{centerZ} - \mathrm{eyeZ})$
$\mathrm{u.x} = \mathrm{upX}, \mathrm{u.y} = \mathrm{upY}, \mathrm{u.z} = \mathrm{upZ}$
$\mathrm{s} = \mathrm{f} \times \mathrm{u},\ \mathrm{u} = \mathrm{s} \times \mathrm{f}$ (cross-product)

3-14-4 Notes

None

3-14-5 Examples

mpLookAt(0, 0, 1, 0, 0, 0, 0, 1, 0); // eye=(0,1,0),up vector is Y axis

3-15 mpVertexPointer

3-15-1 C Specification

void mpVertexPointer(float *pointer)

3-15-2 Parameters

Parameter	Description
Pointer	A pointer to the first coordinates of the first vertex in the array.

3-15-3 Description

mpVertexPointer specifies the location of vertex data.

3-15-4 Notes

float array is only supported. A vertex is constructed with 3 float elements(x,y,z, w is not including).

3-15-5 Examples

```
float vtx_array[] = {
// face 0
//     v0
0.147587,
0.067877,
0.065864,
:
};

mpVertexPointer(3, vtx_array);
```

３-１６mpDrawArrays

３-１６-１C Specification

void mpDrawArrays(int count);

３-１６-２Parameters

Parameter	Description
count	The number of triangle indices of the arrays. For example, 2 triangles would be 6

３-１６-３Description

mpDrawArrays starts rendering. In mpDrawArrays, the IP Core reads the number of counts triangle vertices from the vertex pointer, which is set by mpVertexPointer.

３-１６-４Notes

mpDrawArrays is a blocking function. The program does not execute the next code until current rendering prosess would be finished.

３-１６-５Examples

```
mpDrawArrays(36);       // render 12 triangles
```

3-17 mpRenderColor

3-17-1 C Specification

void mpRenderColor(float red, float green, float blue);

3-17-2 Parameters

Parameter	Description
red,green,blue	red, green and blue values for the wire-frame line color.

3-17-3 Description

mpRenderColor specifies line color for the wire-frame rendering. Each elements of the floating fomat color value is converted to 8-bit color value. The color operation is:

```
r = (int)(red*255.0);
g = (int)(green*255.0);
b = (int)(blue*255.0);

color_8bit =    ((r & 0x3) << 6) |
                ((g & 0x7) << 3)|
                (b & 0x7);              // rgb = 2:3:3
```

3-17-4 Notes

8-bit color is only supported. Each RGB bits of a pixel is 2:3:3.

3-17-5 Examples

```
mpRenderColor(1.0,0.0,0.0);     // pixel color = RED
```

３-１８mpRenderColorU

３-１８-１C Specification

void mpRenderColorU(int color);

３-１８-２Parameters

Parameter	Description
color	8-bit color value for the wire-frame line color.

３-１８-３Description

mpRenderColorU specifies 8-bit line color for wire-frame line rendering.

３-１８-４Notes

8-bit color and the bit assign of RGB elements are system dependent.

３-１８-５Examples

```
// When the system mapped 8-bit color as RGB=2:3:3
mpRenderColorU(0xff);   // white
mpRenderColorU(0xc0);   // red
mpRenderColorU(0x38);   // blue
mpRenderColorU(0x07);   // green
```

３-１９mpClearColor

３-１９-１C Specification

void mpClearColor(float red, float green, float blue)

３-１９-２Parameters

Parameter	Description
red, green, blue	red, green and blue values used when the frame buffer is cleared.

３-１９-３Description

mpClearColor specifies the red, green and blue values used by mpClear to clear the frame buffer (back buffer).

３-１９-４Notes

The 3D Graphics IP Core does not have any buffer clear feature.

３-１９-５Examples

```
mpClearColor(0.5, 0, 0.4);     // R=0.5, G=0, B=0.4
```

3-20 mpClear

3-20-1 C Specification

void mpClear(void)

3-20-2 Parameters

None

3-20-3 Description

mpClear fills the frame buffer (back buffer) by the color specified by mpClearColor.

3-20-4 Notes

The IP Core does not have any buffer clear feature. mpClear() would be implementaed by the system prosessor or system DMAC.

3-20-5 Examples

```
mpClear();      // clear current(front) frame buffer
```

３-２ １ mpSwapBuffers

３-２ １-１ C Specification

void mpSwapBuffers(void)

３-２ １-２ Parameters

None

３-２ １-３ Description

mpSwapBuffers exchanges front and back buffers.

３-２ １-４ Notes

Frame buffer top address register of the IP Core is set front or back in mpSwapBuffers.

３-２ １-５ Examples

mpSwapBuffers();

4 Examples

4 - 1 Simple Cube Rendering

```
float triangle[] = {
   // front
   0.25,   0.25, 0.25,   -0.25,   0.25, 0.25, -0.25, -0.25, 0.25, // triangle0
    0.25,   0.25, 0.25,   -0.25, -0.25, 0.25, 0.25, -0.25, 0.25, // triangle1
   // top
    0.25,   0.25, 0.0,    -0.25,   0.25, 0.0,   -0.25,   0.25, 0.25,   // triangle0
       0.25,   0.25, 0.0,   -0.25,   0.25, 0.25,    0.25,   0.25, 0.25,   // // triangle1
   // bottom
    0.25, -0.25, 0.25,    -0.25, -0.25, 0.25,   -0.25, -0.25, 0,   // triangle0
   0.25, -0.25, 0.25,   -0.25, -0.25, 0, 0.25, -0.25, 0,   // // triangle1
   // left
   -0.25, 0.25, 0.25,   -0.25, 0.25, 0,   -0.25, -0.25, 0,   // triangle0
   -0.25, 0.25, 0.25,    -0.25, -0.25, 0,   -0.25, -0.25, 0.25, // triangle1
   // right
   0.25, 0.25, 0,   0.25, 0.25, 0.25,0.25, -0.25, 0.25,   // triangle0
   0.25, 0.25, 0,   0.25, -0.25, 0.25, 0.25, -0.25, 0, // triangle1
   // back
   -0.25, 0.25, 0,   0.25, 0.25, 0, 0.25, -0.25, 0,   // triangle0
   -0.25, 0.25, 0,    0.25, -0.25, 0,   -0.25, -0.25, 0// triangle1
};

void mp_loop() {
   int i;
   int frame   = 0;

   mpClearColor(0.1, 0.1, 0.1, 1.0);
   mpViewport(0, 0, 640, 480);
   mpMatrixMode(MP_PROJECTION);
   mpPerspective(30.0, 4.0 / 3.0, 1, 100);
   mpVertexPointer(4, triangle);

   while(1) {
      for (i = 0; i <360; i++) {
         printf("frame %d¥n",frame++);

         mpClear();
         mpMatrixMode(MP_MODELVIEW);
         mpLoadIdentity();
         mpLookAt(0, 0, 1, 0, 0, 0, 0, 1, 0);
         mpTranslate(0, 0, -2);
         mpRotate(i, 1, 0, 0);
         mpRotate(i, 0, 0, 1);

         mpDrawArrays( 36);

         mpSwapBuffers ();
      }
   }
}
```

```
int main() {
   mpInit();
   mp_loop();

   return 0;

}
```