Spreadtrum Graphics Interface

1. SGL坐标系统及投影系统
2. 坐标系统规定

坐标系统采用右手坐标系，即以手机屏幕左上角为坐标原点，X轴从左向右延伸，Y轴由上向下延伸，Z轴由屏幕外向屏幕里延伸。

1. 投影系统规定

投影平面与屏幕重合；

投影平面的左上角与坐标系原点重合；

相机可以放置在屏幕外（Z小于零）的任意位置，如下图所示。

被投影图形放置在屏幕内（Z大于等于零）的任意位置。如果被投影图形放置在投影平面上（所有顶点的z值均为零），则相当于2D绘图，相机位置被忽略。



投影示意图

1. Scene（场景）

定义了三维场景中相机的参数，相机参数一旦确定，投影系统就确定。

typedef struct

{

SGL\_CAMERA\_T camera;

}SGL\_SCENE\_T;

typedef struct

{

SGL\_VERTEX3\_T pos;

}SGL\_CAMERA\_T;

注意：如果是2D绘图，则不需要关心场景。

1. Canvas（画布）

用于接收二维图像的对象，定义了图像大小、图像数据格式及存储区域，数据格式支持ARGB888、RGB565和GRAY格式。

投影平面上的图像只能输出到画布上。

用于描述画布属性的结构如下所示：

typedef struct

{

uint32 width;

uint32 height;

void \*mem\_ptr;

uint32 mem\_width;

SGL\_DATA\_FORMAT\_E data\_format;

SGL\_RECT2\_T draw\_rect;

}SGL\_CANVAS\_T;

width：图像高度；

height：图像宽度；

mem\_ptr：图像数据对应的存储区域；

mem\_width：图像每个存储行所占的内存宽度，以字节为单位；

data\_format：图像数据格式。

draw\_rect：绘制区域，落在该区域内的内容将被绘制，区域外的内容将被裁剪。

1. Shape（形状）

对空间图形的抽象，包括直线、矩形平面、多边形等等。每种Shape对应一个属性结构体，用于描述该Shape的属性。暂时定义了三种Shape。

* SGL\_SHAPE\_LINE

直线，对应的属性结构体如下：

typedef struct

{

SGL\_VERTEX3\_T begin;

SGL\_VERTEX3\_T end;

SGL\_COLOR\_ARGB\_T color;

SGL\_LINE\_STYLE\_E style;

SGL\_LINE\_END\_CAP\_E end\_cap;

int32 width;

}SGL\_SHAPE\_LINE\_T;

begin：起点；

end：终点；

color：直线颜色；

style：线型；

end\_cap：线端

width：线宽

* SGL\_SHAPE\_POLYGON

多边形，对应的属性结构体如下：

typedef struct

{

uint32 vertex\_num;

SGL\_VERTEX3\_T \*vertex\_ptr;

SGL\_POLYGON\_STYLE\_E style;

int32 outline\_width;

union

{

SGL\_COLOR\_RGB888\_T outline\_color;

SGL\_COLOR\_RGB888\_T fill\_color;

SGL\_COLOR\_RGB888\_T gradient\_color[2];

}color;

}SGL\_SHAPE\_POLYGON\_T;

vertex\_num：多边形顶点个数；

vertex\_ptr：多边形顶点指针；

style：多边形风格，包括如下风格

|  |  |
| --- | --- |
| SGL\_PGN\_SOLID\_OUTLINE | 轮廓多边形 |
| SGL\_PGN\_FILL | 单色填充多边形 |
| SGL\_PGN\_GRADIENT\_FILL\_L2R | 渐变填充多边形，颜色从左至右渐变 |
| SGL\_PGN\_GRADIENT\_FILL\_T2B | 渐变填充多边形，颜色从上至下渐变 |

outline\_width：轮廓线宽；

color.outline\_color：轮廓多边形的线宽；

color.fill\_color：单色填充多边表的颜色；

color.gradient\_color[2]：渐变填充多边形的颜色，color.gradient\_color[0]为起始颜色color.gradient\_color[1]为终止颜色。

1. Transform

坐标变换包括平移、旋转等操作，在投影变换之前根据变换参数对图形坐标进行变换。

绘制图形时可以通过SGL\_TRANSFORM\_T结构设置变换的详细参数。SGL\_TRANSFORM\_T结构定义如下：

typedef struct

{

SGL\_TRANSFORM\_TYPE\_E type;

union

{

SGL\_MOVE\_T move;

SGL\_ROTATE\_T rotate;

} op\_param;

}SGL\_TRANSFORM\_T;

type：变换类型，定义了平移和旋转两种类型；

op：变换的属性参数，根据变换类型设置对应的属性参数；

绘制图形时，在绘制参数中设置变换次数和变换属性数组

typedef struct

{

void \*shape\_ptr;

SGL\_HANDLE\_T canvas\_handle;

int32 trans\_count;

SGL\_TRANSFORM\_T \*trans\_array\_ptr;

}SGL\_SHAPE\_DRAW\_IN\_T;

shape\_ptr：图形参数，每种图形一个对应的属性结构体；

canvas\_handle：画布handle；

trans\_count：变换次数，如设置为0，则不进行变换操作；

trans\_array\_ptr：变换属性数组，大小与变换次数一致；如设置为NULL，则不进行变换操作。

注意：变换顺序会影响变换结果。

1. API

主要包括三组API：

1. Canvas相关API
2. 创建画布

PUBLIC SGL\_HANDLE\_T SGL\_CreateCanvas(SGL\_CANVAS\_T \*canvas\_ptr);

1. 用指定颜色清空画布

PUBLIC void SGL\_ClearCanvas(SGL\_HANDLE\_T canvas\_handle, uint32 clear\_color);

1. 销毁画布

PUBLIC void SGL\_DestroyCanvas(SGL\_HANDLE\_T canvas\_handle);

1. Shape相关API
2. 创建指定类型的Shape

PUBLIC SGL\_HANDLE\_T SGL\_CreateShape(SGL\_SHAPE\_TYPE\_E shape\_type,

SGL\_HANDLE\_T scene\_handle);

注意：scene\_handle可以设置为NULL表示不关心场景（2D绘图）或取默认的场景参数。

1. 绘制Shape，投影结果将输出到指定的Canvas上

PUBLIC int32 SGL\_DrawShape(SGL\_HANDLE\_T shape\_handle,

SGL\_DRAW\_SHAPE\_IN\_T \*draw\_shape\_in\_ptr,

SGL\_DRAW\_SHAPE\_OUT\_T \*draw\_shape\_out\_ptr);

SGL\_DRAW\_SHAPE\_IN\_T结构为绘制形状所需的输入信息，定义如下：

typedef struct

{

void \*shape\_ptr;

SGL\_HANDLE\_T canvas\_handle;

int32 trans\_count;

SGL\_TRANSFORM\_T \*trans\_array\_ptr;

}SGL\_SHAPE\_DRAW\_IN\_T;

shape\_ptr：图形参数，每种图形一个对应的属性结构体；

canvas\_handle：画布handle；

trans\_count：变换次数，如设置为0，则不进行变换操作；

trans\_array\_ptr：变换属性数组，大小与变换次数一致；如设置为NULL，则不进行变换操作。

SGL\_DRAW\_SHAPE\_OUT\_T为绘制图形后的输出参数，定义如下所示：

typedef struct

{

SGL\_RECT2\_T update\_rect;

}SGL\_DRAW\_SHAPE\_OUT\_T;

update\_rect：需要更新的区域，以画布的左上角点为坐标原点

注意：draw\_shape\_out\_ptr可以设置为NULL表示不关心输出参数。

1. 销毁Shape

PUBLIC void SGL\_DestroyShape(SGL\_HANDLE\_T shape\_handle);

1. Scene相关API
2. 创建场景

PUBLIC SGL\_HANDLE\_T SGL\_CreateScene(SGL\_SCENE\_T \*scene\_ptr);

1. 销毁场景

PUBLIC int32 SGL\_DestroyScene(SGL\_HANDLE\_T scene\_handle);

注意：如果是2D绘图或者取默认场景，则可以忽略该组函数。

1. 2D绘图一般操作流程



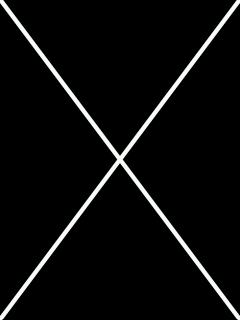
1. 3D绘图一般操作流程

三维绘图与二维绘图过程大致相同，唯一的区别是绘图之前需要创建场景对象，绘图结束后销毁场景对象。



1. SampleCode
2. 直线绘制

TEST\_DrawLine函数在一张240X320的画布上绘制两条交叉的线。画布为黑色，直线为白色。绘制结果如图所示：



代码如下：

PUBLIC void TEST\_DrawLine(const char \*save\_name)

{

SGL\_HANDLE\_T canvas\_handle = NULL;

SGL\_CANVAS\_T canvas = {0};

SGL\_HANDLE\_T shape\_handle = NULL;

SGL\_DRAW\_SHAPE\_IN\_T draw\_shape\_in = {0};

SGL\_DRAW\_SHAPE\_OUT\_T draw\_shape\_out = {0};

SGL\_SHAPE\_LINE\_T line = {0};

////////////////////////////////////////////////////////////

//create canvas

canvas.data\_format = SGL\_FORMAT\_RGB565;

canvas.width = 240;

canvas.height = 320;

canvas.mem\_width = 240;

canvas.mem\_ptr = SGL\_ALLOC(canvas.mem\_width \* canvas.height \* 2);

if (NULL == canvas.mem\_ptr)

{

goto TEST\_EXIT;

}

SCI\_MEMSET(canvas.mem\_ptr, 0x0, canvas.mem\_width \* canvas.height);

canvas\_handle = SGL\_CreateCanvas(&canvas);

if (NULL == canvas\_handle)

{

goto TEST\_EXIT;

}

////////////////////////////////////////////////////////////

//draw shape

shape\_handle = SGL\_CreateShape(SGL\_SHAPE\_LINE, NULL);

if (NULL == shape\_handle)

{

goto TEST\_EXIT;

}

draw\_shape\_in.canvas\_handle = canvas\_handle;

draw\_shape\_in.shape\_ptr = (void \*)&line;

/\*line property\*/

line.begin.x = 0;

line.begin.y = 0;

line.begin.z = 0;

line.end.x = 239;

line.end.y = 319;

line.end.z = 0;

line.color = 0xFFFF;

line.width = 4;

line.style = SGL\_LINE\_SOLID;

if (SGL\_SUCCESS != SGL\_DrawShape(shape\_handle, &draw\_shape\_in,

&draw\_shape\_out))

{

goto TEST\_EXIT;

}

/\*line property\*/

line.begin.x = 239;

line.begin.y = 0;

line.begin.z = 0;

line.end.x = 0;

line.end.y = 319;

line.end.z = 0;

if (SGL\_SUCCESS != SGL\_DrawShape(shape\_handle, &draw\_shape\_in,

&draw\_shape\_out))

{

goto TEST\_EXIT;

}

////////////////////////////////////////////////////////////

//save canvas to bitmap for debug

if (save\_name)

{

SaveCanvas(&canvas, save\_name);

}

////////////////////////////////////////////////////////////

//destroy resoures

TEST\_EXIT:

SGL\_DestroyCanvas(canvas\_handle);

SGL\_DestroyShape(shape\_handle);

if (NULL != canvas.mem\_ptr)

{

SGL\_FREE(canvas.mem\_ptr);

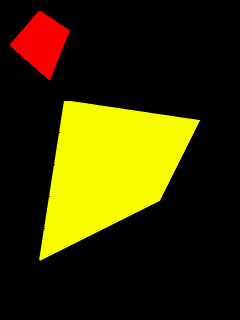
}

return;

}

1. 多边形填充

TEST\_DrawPolygon函数在一张240X320的画布上绘制两个多边形。画布为黑色，多边形一个为红色，一个为黑色。绘制结果如图所示：



PUBLIC void TEST\_DrawPolygon(const char \*save\_name)

{

SGL\_SCENE\_T scene = {0};

SGL\_HANDLE\_T canvas\_handle = NULL;

SGL\_CANVAS\_T canvas = {0};

SGL\_HANDLE\_T shape\_handle = NULL;

SGL\_DRAW\_SHAPE\_IN\_T draw\_shape\_in = {0};

SGL\_DRAW\_SHAPE\_OUT\_T draw\_shape\_out = {0};

SGL\_SHAPE\_POLYGON\_T polygon = {0};

SGL\_VERTEX3\_T vertex[4] = {0};

////////////////////////////////////////////////////////////

//create canvas

canvas.data\_format = SGL\_FORMAT\_RGB565;

canvas.width = 240;

canvas.height = 320;

canvas.mem\_width = 240;

canvas.draw\_rect.left = 0;

canvas.draw\_rect.top = 0;

canvas.draw\_rect.right = canvas.width - 1;

canvas.draw\_rect.bottom = canvas.height - 1;

canvas.mem\_ptr = SGL\_ALLOC(canvas.mem\_width \* canvas.height \* 2);

if (NULL == canvas.mem\_ptr)

{

goto TEST\_EXIT;

}

canvas\_handle = SGL\_CreateCanvas(&canvas);

if (NULL == canvas\_handle)

{

goto TEST\_EXIT;

}

SGL\_ClearCanvas(canvas\_handle, 0);

////////////////////////////////////////////////////////////

//draw shape

shape\_handle = SGL\_CreateShape(SGL\_SHAPE\_POLYGON, NULL);

if (NULL == shape\_handle)

{

goto TEST\_EXIT;

}

draw\_shape\_in.canvas\_handle = canvas\_handle;

draw\_shape\_in.shape\_ptr = (void \*)&polygon;

/\*polygon property\*/

polygon.color.fill\_color = 0xff0000;

polygon.style = SGL\_PGN\_FILL;

polygon.vertex\_num = 4;

polygon.vertex\_ptr = vertex;

vertex[0].x = 40;

vertex[0].y = 10;

vertex[1].x = 70;

vertex[1].y = 30;

vertex[2].x = 50;

vertex[2].y = 80;

vertex[3].x = 10;

vertex[3].y = 45;

if (SGL\_SUCCESS != SGL\_DrawShape(shape\_handle, &draw\_shape\_in, &draw\_shape\_out))

{

goto TEST\_EXIT;

}

/\*line property\*/

polygon.color.fill\_color = 0xffff00;

polygon.style = SGL\_PGN\_FILL;

polygon.vertex\_num = 4;

polygon.vertex\_ptr = vertex;

vertex[0].x = 65;

vertex[0].y = 100;

vertex[1].x = 200;

vertex[1].y = 120;

vertex[2].x = 160;

vertex[2].y = 200;

vertex[3].x = 40;

vertex[3].y = 260;

if (SGL\_SUCCESS != SGL\_DrawShape(shape\_handle, &draw\_shape\_in, &draw\_shape\_out))

{

goto TEST\_EXIT;

}

////////////////////////////////////////////////////////////

//save canvas to bitmap for debug

if (save\_name)

{

SaveCanvas(&canvas, save\_name);

}

////////////////////////////////////////////////////////////

//destroy resoures

TEST\_EXIT:

SGL\_DestroyCanvas(canvas\_handle);

SGL\_DestroyShape(shape\_handle);

if (NULL != canvas.mem\_ptr)

{

SGL\_FREE(canvas.mem\_ptr);

}

return;

}