Android 在 HAL 中提供了gralloc 模块，封装了用户层对帧缓冲设备的所有操作接口，并通过 SurfaceFlinger 服务向应用提供显示支持。在启动过程中,系统会加载 gralloc 模块，然后打开帧缓冲设备，获取设备的各种参数并完成 gralloc 模块的初始化。当应用程序需要把内容显示到 屏幕上时，需要通过 gralloc 模块申请一块图形缓冲区，然后将这块图形缓冲区映射到自己的地址空间并写入内容即可。当应用程序不再需要这块图形缓冲区时需要通过 gralloc 模块释放掉，然后解除对缓冲区的映射。

**gralloc\_drm\_intel.c 程序分析：**

gralloc\_drm\_intel.c实现了gralloc\_drm模块调用intel集显时的一些操作，包括获取集显信息、创建gralloc缓冲设备、创建intel图形缓冲区、刷新intel内存缓冲区、销毁intel内存缓冲区等。

1、首先定义结构体intel\_info用来存储需要调用intel集显时的一些信息，定义结构体intel\_buffer用来存储intel集显的缓冲信息

2、下一个批处理的intel集显控制信息

static int batch\_next(struct intel\_info \*info)

{

info->cur = info->batch;

if (info->batch\_ibo)

drm\_intel\_bo\_unreference(info->batch\_ibo);

info->batch\_ibo = drm\_intel\_bo\_alloc(info->bufmgr,

"gralloc-batchbuffer", info->size, 4096);

return (info->batch\_ibo) ? 0 : -ENOMEM;

}

3、统计需要批处理的intel集显信息

static int batch\_count(struct intel\_info \*info)

{

return info->cur - info->batch;

}

4、需要批处理的下一个intel集显信息

static void batch\_dword(struct intel\_info \*info, uint32\_t dword)

{

\*info->cur++ = dword;

}

5、

static int batch\_reloc(struct intel\_info \*info, struct gralloc\_drm\_bo\_t \*bo,

uint32\_t read\_domains, uint32\_t write\_domain)

{

struct intel\_buffer \*target = (struct intel\_buffer \*) bo;

uint32\_t offset = (info->cur - info->batch) \* sizeof(info->batch[0]);

int ret;

ret = drm\_intel\_bo\_emit\_reloc(info->batch\_ibo, offset,

target->ibo, 0, read\_domains, write\_domain);

if (!ret)

batch\_dword(info, target->ibo->offset);

return ret;

}

6、批量刷新当前intel集显信息

static int batch\_flush(struct intel\_info \*info)

batch\_flush(struct intel\_info \*info)

{

int size, ret;

batch\_dword(info, MI\_BATCH\_BUFFER\_END);

size = batch\_count(info);

if (size & 1) {

batch\_dword(info, MI\_NOOP);

size = batch\_count(info);

}

size \*= sizeof(info->batch[0]);

ret = drm\_intel\_bo\_subdata(info->batch\_ibo, 0, size, info->batch);

if (ret) {

ALOGE("failed to subdata batch");

goto fail;

}

ret = drm\_intel\_bo\_mrb\_exec(info->batch\_ibo, size,

NULL, 0, 0, info->exec\_blt);

if (ret) {

ALOGE("failed to exec batch");

goto fail;

}

return batch\_next(info);

fail:

info->cur = info->batch;

return ret;

}

7、批量处理预留的intel集显信息

static int batch\_reserve(struct intel\_info \*info, int count)

{

int ret = 0;

if (batch\_count(info) + count > info->capacity)

ret = batch\_flush(info);

return ret;

}

8、批量销毁intel集显信息

static void batch\_destroy(struct intel\_info \*info)

{

if (info->batch\_ibo) {

drm\_intel\_bo\_unreference(info->batch\_ibo);

info->batch\_ibo = NULL;

}

if (info->batch) {

free(info->batch);

info->batch = NULL;

}

}

9、批量初始化intel集显信息

static int batch\_init(struct intel\_info \*info)

{

int ret;

info->capacity = 512;

info->size = (info->capacity + 16) \* sizeof(info->batch[0]);

info->batch = malloc(info->size);

if (!info->batch)

return -ENOMEM;

ret = batch\_next(info);

if (ret) {

free(info->batch);

info->batch = NULL;

}

return ret;

}

10、定义intel集显的图像分解格式

static void intel\_resolve\_format(struct gralloc\_drm\_drv\_t \*drv,

struct gralloc\_drm\_bo\_t \*bo,

uint32\_t \*pitches, uint32\_t \*offsets, uint32\_t \*handles)

{

/\*

\* TODO - should take account hw specific padding, alignment

\* for camera, video decoder etc.

\*/

struct intel\_buffer \*ib = (struct intel\_buffer \*) bo;

memset(pitches, 0, 4 \* sizeof(uint32\_t));

memset(offsets, 0, 4 \* sizeof(uint32\_t));

memset(handles, 0, 4 \* sizeof(uint32\_t));

pitches[0] = ib->base.handle->stride;

handles[0] = ib->base.fb\_handle;

switch(ib->base.handle->format) {

case HAL\_PIXEL\_FORMAT\_YV12:

// U and V stride are half of Y plane

pitches[2] = pitches[0]/2;

pitches[1] = pitches[0]/2;

// like I420 but U and V are in reverse order

offsets[2] = offsets[0] +

pitches[0] \* ib->base.handle->height;

offsets[1] = offsets[2] +

pitches[2] \* ib->base.handle->height/2;

handles[1] = handles[2] = handles[0];

break;

case HAL\_PIXEL\_FORMAT\_DRM\_NV12:

// U and V are interleaved in 2nd plane

pitches[1] = pitches[0];

offsets[1] = offsets[0] +

pitches[0] \* ib->base.handle->height;

handles[1] = handles[0];

break;

}

}

11、定义intel集显的图像位块传输方法

static void intel\_blit(struct gralloc\_drm\_drv\_t \*drv,

struct gralloc\_drm\_bo\_t \*dst, //

struct gralloc\_drm\_bo\_t \*src,

uint16\_t dst\_x1, uint16\_t dst\_y1,

uint16\_t dst\_x2, uint16\_t dst\_y2,

uint16\_t src\_x1, uint16\_t src\_y1,

uint16\_t src\_x2, uint16\_t src\_y2)

{

struct intel\_info \*info = (struct intel\_info \*) drv;

struct intel\_buffer \*dst\_ib = (struct intel\_buffer \*) dst;

struct intel\_buffer \*src\_ib = (struct intel\_buffer \*) src;

drm\_intel\_bo \*bo\_table[3];

uint32\_t cmd, br13, dst\_pitch, src\_pitch;

/\*

\* XY\_SRC\_COPY\_BLT\_CMD does not support scaling,

\* rectangle dimensions much match

\*/

if (src\_x2 - src\_x1 != dst\_x2 - dst\_x1 ||

src\_y2 - src\_y1 != dst\_y2 - dst\_y1) {

ALOGE("%s, src and dst rect must match", \_\_func\_\_);

return;

}

if (dst->handle->format != src->handle->format) {

ALOGE("%s, src and dst format must match", \_\_func\_\_);

return;

}

/\* nothing to blit \*/

if (src\_x2 <= src\_x1 || src\_y2 <= src\_y1)

return;

/\* clamp x2, y2 to surface size \*/

if (src\_x2 > src->handle->width)

src\_x2 = src->handle->width;

if (src\_y2 > src->handle->height)

src\_y2 = src->handle->height;

if (dst\_x2 > dst->handle->width)

dst\_x2 = dst->handle->width;

if (dst\_y2 > dst->handle->height)

dst\_y2 = dst->handle->height;

bo\_table[0] = info->batch\_ibo;

bo\_table[1] = src\_ib->ibo;

bo\_table[2] = dst\_ib->ibo;

if (drm\_intel\_bufmgr\_check\_aperture\_space(bo\_table, 3)) {

if (batch\_flush(info))

return;

assert(!drm\_intel\_bufmgr\_check\_aperture\_space(bo\_table, 3));

}

cmd = XY\_SRC\_COPY\_BLT\_CMD;

br13 = 0xcc << 16; /\* ROP\_S/GXcopy \*/

dst\_pitch = dst->handle->stride;

src\_pitch = src->handle->stride;

/\* Blit pitch must be dword-aligned. Otherwise, the hardware appears to

\* drop the low bits.

\*/

if (src\_pitch % 4 != 0 || dst\_pitch % 4 != 0) {

ALOGE("%s, src and dst pitch must be dword aligned", \_\_func\_\_);

return;

}

switch (gralloc\_drm\_get\_bpp(dst->handle->format)) {

case 1:

break;

case 2:

br13 |= (1 << 24);

break;

case 4:

br13 |= (1 << 24) | (1 << 25);

cmd |= XY\_SRC\_COPY\_BLT\_WRITE\_ALPHA | XY\_SRC\_COPY\_BLT\_WRITE\_RGB;

break;

default:

ALOGE("%s, copy with unsupported format", \_\_func\_\_);

return;

}

if (info->gen >= 40) {

if (dst\_ib->tiling != I915\_TILING\_NONE) {

assert(dst\_pitch % 512 == 0);

dst\_pitch >>= 2;

cmd |= XY\_SRC\_COPY\_BLT\_DST\_TILED;

}

if (src\_ib->tiling != I915\_TILING\_NONE) {

assert(src\_pitch % 512 == 0);

src\_pitch >>= 2;

cmd |= XY\_SRC\_COPY\_BLT\_SRC\_TILED;

}

}

if (batch\_reserve(info, 8))

return;

batch\_dword(info, cmd);

batch\_dword(info, br13 | (uint16\_t)dst\_pitch);

batch\_dword(info, (dst\_y1 << 16) | dst\_x1);

batch\_dword(info, (dst\_y2 << 16) | dst\_x2);

batch\_reloc(info, dst, I915\_GEM\_DOMAIN\_RENDER, I915\_GEM\_DOMAIN\_RENDER);

batch\_dword(info, (src\_y1 << 16) | src\_x1);

batch\_dword(info, (uint16\_t)src\_pitch);

batch\_reloc(info, src, I915\_GEM\_DOMAIN\_RENDER, 0);

if (info->gen >= 60) {

batch\_reserve(info, 4);

batch\_dword(info, MI\_FLUSH\_DW | 2);

batch\_dword(info, 0);

batch\_dword(info, 0);

batch\_dword(info, 0);

}

else {

int flags = (info->gen >= 40) ? 0 :

MI\_WRITE\_DIRTY\_STATE | MI\_INVALIDATE\_MAP\_CACHE;

batch\_reserve(info, 1);

batch\_dword(info, MI\_FLUSH | flags);

}

batch\_flush(info);

}

12、为intel集显分配gralloc缓冲设备

static drm\_intel\_bo \*alloc\_ibo(struct intel\_info \*info,

const struct gralloc\_drm\_handle\_t \*handle,

uint32\_t \*tiling, unsigned long \*stride)

{

drm\_intel\_bo \*ibo;

const char \*name;

int aligned\_width, aligned\_height, bpp;

unsigned long flags;

flags = 0;

bpp = gralloc\_drm\_get\_bpp(handle->format);

if (!bpp) {

ALOGE("unrecognized format 0x%x", handle->format);

return NULL;

}

aligned\_width = handle->width;

aligned\_height = handle->height;

gralloc\_drm\_align\_geometry(handle->format,

&aligned\_width, &aligned\_height);

if (handle->usage & GRALLOC\_USAGE\_HW\_FB) {

unsigned long max\_stride;

max\_stride = 32 \* 1024;

if (info->gen < 50)

max\_stride /= 2;

if (info->gen < 40)

max\_stride /= 2;

name = "gralloc-fb";

aligned\_width = ALIGN(aligned\_width, 64);

flags = BO\_ALLOC\_FOR\_RENDER;

\*tiling = I915\_TILING\_X;

\*stride = aligned\_width \* bpp;

if (\*stride > max\_stride) {

\*tiling = I915\_TILING\_NONE;

max\_stride = 32 \* 1024;

if (\*stride > max\_stride)

return NULL;

}

while (1) {

ibo = drm\_intel\_bo\_alloc\_tiled(info->bufmgr, name,

aligned\_width, aligned\_height,

bpp, tiling, stride, flags);

if (!ibo || \*stride > max\_stride) {

if (ibo) {

drm\_intel\_bo\_unreference(ibo);

ibo = NULL;

}

if (\*tiling != I915\_TILING\_NONE) {

/\* retry \*/

\*tiling = I915\_TILING\_NONE;

max\_stride = 32 \* 1024;

continue;

}

}

if (ibo)

drm\_intel\_bo\_disable\_reuse(ibo);

break;

}

}

else {

if (handle->usage & (GRALLOC\_USAGE\_SW\_READ\_OFTEN |

GRALLOC\_USAGE\_SW\_WRITE\_OFTEN))

\*tiling = I915\_TILING\_NONE;

else if ((handle->usage & GRALLOC\_USAGE\_HW\_RENDER) ||

((handle->usage & GRALLOC\_USAGE\_HW\_TEXTURE) &&

handle->width >= 64))

\*tiling = I915\_TILING\_X;

else

\*tiling = I915\_TILING\_NONE;

if (handle->usage & GRALLOC\_USAGE\_HW\_TEXTURE) {

name = "gralloc-texture";

/\* see 2D texture layout of DRI drivers \*/

aligned\_width = ALIGN(aligned\_width, 4);

aligned\_height = ALIGN(aligned\_height, 2);

}

else {

name = "gralloc-buffer";

}

if (handle->usage & GRALLOC\_USAGE\_HW\_RENDER)

flags = BO\_ALLOC\_FOR\_RENDER;

ibo = drm\_intel\_bo\_alloc\_tiled(info->bufmgr, name,

aligned\_width, aligned\_height,

bpp, tiling, stride, flags);

}

return ibo;

}

13、实现为intel集显分配图形缓冲区

static struct gralloc\_drm\_bo\_t \*intel\_alloc(struct gralloc\_drm\_drv\_t \*drv,

struct gralloc\_drm\_handle\_t \*handle)

{

struct intel\_info \*info = (struct intel\_info \*) drv;

struct intel\_buffer \*ib;

ib = calloc(1, sizeof(\*ib));

if (!ib)

return NULL;

if (handle->name) {

uint32\_t dummy;

ib->ibo = drm\_intel\_bo\_gem\_create\_from\_name(info->bufmgr,

"gralloc-r", handle->name);

if (!ib->ibo) {

ALOGE("failed to create ibo from name %u",

handle->name);

free(ib);

return NULL;

}

if (drm\_intel\_bo\_get\_tiling(ib->ibo, &ib->tiling, &dummy)) {

ALOGE("failed to get ibo tiling");

drm\_intel\_bo\_unreference(ib->ibo);

free(ib);

return NULL;

}

}

else {

unsigned long stride;

ib->ibo = alloc\_ibo(info, handle, &ib->tiling, &stride);

if (!ib->ibo) {

ALOGE("failed to allocate ibo %dx%d (format %d)",

handle->width,

handle->height,

handle->format);

free(ib);

return NULL;

}

handle->stride = stride;

if (drm\_intel\_bo\_flink(ib->ibo, (uint32\_t \*) &handle->name)) {

ALOGE("failed to flink ibo");

drm\_intel\_bo\_unreference(ib->ibo);

free(ib);

return NULL;

}

}

ib->base.fb\_handle = ib->ibo->handle;

ib->base.handle = handle;

return &ib->base;

}

14、释放intel集显的图形缓冲区

static void intel\_free(struct gralloc\_drm\_drv\_t \*drv,

struct gralloc\_drm\_bo\_t \*bo)

{

struct intel\_buffer \*ib = (struct intel\_buffer \*) bo;

drm\_intel\_bo\_unreference(ib->ibo);

free(ib);

}

15、将intel集显地址空间映射到图形缓冲区

static int intel\_map(struct gralloc\_drm\_drv\_t \*drv,

struct gralloc\_drm\_bo\_t \*bo,

int x, int y, int w, int h,

int enable\_write, void \*\*addr)

{

struct intel\_buffer \*ib = (struct intel\_buffer \*) bo;

int err;

if (ib->tiling != I915\_TILING\_NONE ||

(ib->base.handle->usage & GRALLOC\_USAGE\_HW\_FB))

err = drm\_intel\_gem\_bo\_map\_gtt(ib->ibo);

else

err = drm\_intel\_bo\_map(ib->ibo, enable\_write);

if (!err)

\*addr = ib->ibo->virtual;

return err;

}

16、取消映射

static void intel\_unmap(struct gralloc\_drm\_drv\_t \*drv,

struct gralloc\_drm\_bo\_t \*bo)

{

struct intel\_buffer \*ib = (struct intel\_buffer \*) bo;

if (ib->tiling != I915\_TILING\_NONE ||

(ib->base.handle->usage & GRALLOC\_USAGE\_HW\_FB))

drm\_intel\_gem\_bo\_unmap\_gtt(ib->ibo);

else

drm\_intel\_bo\_unmap(ib->ibo);

}

17、初始化intel的kms特征

static void intel\_init\_kms\_features(struct gralloc\_drm\_drv\_t \*drv,

struct gralloc\_drm\_t \*drm)

{

struct intel\_info \*info = (struct intel\_info \*) drv;

struct drm\_i915\_getparam gp;

int pageflipping, id, has\_blt;

switch (drm->primary.fb\_format) {

case HAL\_PIXEL\_FORMAT\_BGRA\_8888:

case HAL\_PIXEL\_FORMAT\_RGB\_565:

break;

default:

drm->primary.fb\_format = HAL\_PIXEL\_FORMAT\_BGRA\_8888;

break;

}

drm->mode\_quirk\_vmwgfx = 0;

/\* why? \*/

drm->mode\_sync\_flip = 1;

memset(&gp, 0, sizeof(gp));

gp.param = I915\_PARAM\_HAS\_PAGEFLIPPING;

gp.value = &pageflipping;

if (drmCommandWriteRead(drm->fd, DRM\_I915\_GETPARAM, &gp, sizeof(gp)))

pageflipping = 0;

memset(&gp, 0, sizeof(gp));

gp.param = I915\_PARAM\_CHIPSET\_ID;

gp.value = &id;

if (drmCommandWriteRead(drm->fd, DRM\_I915\_GETPARAM, &gp, sizeof(gp)))

id = 0;

memset(&gp, 0, sizeof(gp));

gp.param = I915\_PARAM\_HAS\_BLT;

gp.value = &has\_blt;

if (drmCommandWriteRead(drm->fd, DRM\_I915\_GETPARAM, &gp, sizeof(gp)))

has\_blt = 0;

info->exec\_blt = has\_blt ? I915\_EXEC\_BLT : 0;

/\* GEN4, G4X, GEN5, GEN6, GEN7 \*/

if ((IS\_9XX(id) || IS\_G4X(id)) && !IS\_GEN3(id)) {

if (IS\_GEN7(id))

info->gen = 70;

else if (IS\_GEN6(id))

info->gen = 60;

else if (IS\_GEN5(id))

info->gen = 50;

else

info->gen = 40;

}

else {

info->gen = 30;

}

if (pageflipping && info->gen > 30)

drm->swap\_mode = DRM\_SWAP\_FLIP;

else if (info->batch && info->gen == 30)

drm->swap\_mode = DRM\_SWAP\_COPY;

else

drm->swap\_mode = DRM\_SWAP\_SETCRTC;

if (drm->resources) {

int pipe;

pipe = drm\_intel\_get\_pipe\_from\_crtc\_id(info->bufmgr,

drm->primary.crtc\_id);

drm->swap\_interval = (pipe >= 0) ? 1 : 0;

drm->vblank\_secondary = (pipe > 0);

}

else {

drm->swap\_interval = 0;

}

}

18、销毁所创建的intel集显驱动图形缓冲区和信息

static void intel\_destroy(struct gralloc\_drm\_drv\_t \*drv)

{

struct intel\_info \*info = (struct intel\_info \*) drv;

batch\_destroy(info);

drm\_intel\_bufmgr\_destroy(info->bufmgr);

free(info);

}

19、调用gralloc\_drm\_drv\_t为intel集显创建驱动

struct gralloc\_drm\_drv\_t \*gralloc\_drm\_drv\_create\_for\_intel(int fd)

{

struct intel\_info \*info;

info = calloc(1, sizeof(\*info));

if (!info) {

ALOGE("failed to allocate driver info");

return NULL;

}

info->fd = fd;

info->bufmgr = drm\_intel\_bufmgr\_gem\_init(info->fd, 16 \* 1024);

if (!info->bufmgr) {

ALOGE("failed to create buffer manager");

free(info);

return NULL;

}

batch\_init(info);

info->base.destroy = intel\_destroy;

info->base.init\_kms\_features = intel\_init\_kms\_features;

info->base.alloc = intel\_alloc;

info->base.free = intel\_free;

info->base.map = intel\_map;

info->base.unmap = intel\_unmap;

info->base.blit = intel\_blit;

info->base.resolve\_format = intel\_resolve\_format;

return &info->base;

}