**[Gallery 3D 分析](http://yueguc.javaeye.com/blog/748924)**

**文章分类:**[**移动开发**](http://www.javaeye.com/blogs/category/mobile)

Gallery 3D 分析   
  
1.Layer 是个抽象类。   
有5个变量，分别是mX;mY（位置坐标）;mWidth;mHeight（宽与高）;mHidden （是否隐藏）;   
一个抽象方法，非抽象子类都有实现。 

**Java代码**

1. **public** **abstract** **void** generate(RenderView view, RenderView.Lists lists);

其他的可以在子类中重写 

**Java代码**

1. // Returns true if something is animating.
2. **public** **boolean** update(RenderView view, **float** frameInterval) {
3. **return** **false**;
4. }
6. **public** **void** renderOpaque(RenderView view, GL11 gl) {
7. }
9. **public** **void** renderBlended(RenderView view, GL11 gl) {
10. }
12. **public** **boolean** onTouchEvent(MotionEvent event) {
13. **return** **false**;
14. }
16. // Allows subclasses to further constrain the hit test defined by layer
17. // bounds.
18. **public** **boolean** containsPoint(**float** x, **float** y) {
19. **return** **true**;
20. }
22. **protected** **void** onSurfaceCreated(RenderView view, GL11 gl) {
23. }
25. **protected** **void** onSizeChanged() {
26. }
28. **protected** **void** onHiddenChanged() {
29. }

有好几个类都继承了Layer抽象类：   
BackgroundLayer   
CanvasLayer   
GridLayer   
HudLayer   
LoadingLayer   
RootLayer   
  
  
其中用到的RenderView类

**Java代码**

1. **final** **class** RenderView **extends** GLSurfaceView **implements** GLSurfaceView.Renderer, SensorEventListener

里面有个静态 final 类 Lists   
里面定义了updateList，opaqueList，blendedList，hitTestList，systemList等以Layer为对象内容的ArrayList;以及一个清空所有ArrayList的方法Clear().

### [Gallery 3D 代码分析之概览](http://yueguc.javaeye.com/blog/749037)

**文章分类:**[**移动开发**](http://www.javaeye.com/blogs/category/mobile)

Gallery 3D UI 非常炫， 如下图所示：

#### 需要明确的几个问题

##### 伪 2D 还是 3D：

gallery3d 基于 android SDK OpenGL ES 接口开发，使用了 Java API,没有使用 NDK。

##### 图片如何显示：

在 OpenGL ES 中，要显示图片，需要定义一个四边形，然后把图片当作 texture 贴到四边形上。

##### 布局及特效如何实现：

这是 gallery3d 的精华所在，需认真分析。

##### 大数据量图片/cache 如何实现和管理：

gallery3d 有缓冲区的设计，非常不错，需要认真分析。

##### 动画引擎：

简单的讲，动画引擎对外可表现为一个接口：

**Java代码**

1. **float** animate(**float** initVal, **float** \*currentVal, **long** timeElapsed, **long** duration)

即，给定初始值(initVal)，动画引擎根据逝去的时间(timeElapsed)和动画总时间（duration）计算下一帧对应的值 (currentVal)，这个值可能是位置坐标，也可能是一个矩阵 matrix，或者是其它的属性。显示一帧就调用该函数更新actor的属性，各个帧连起来显示就成了动画。

##### 3D坐标与2D坐标的转换：

这个需要仔细分析。

##### 使用缓冲区对象 (GL\_OES\_vertex\_buffer\_object)

OpenGL ES 中的顶点数组使得几何图元的显示方便了很多，但是如果每次都要向　OPENGL　发送一大块数据，而这数据其实并没有修改过，那么这传输就是冗余的。所以 这里添加了缓冲区对象，将顶点数组存储在服务器端的缓冲区对象中。

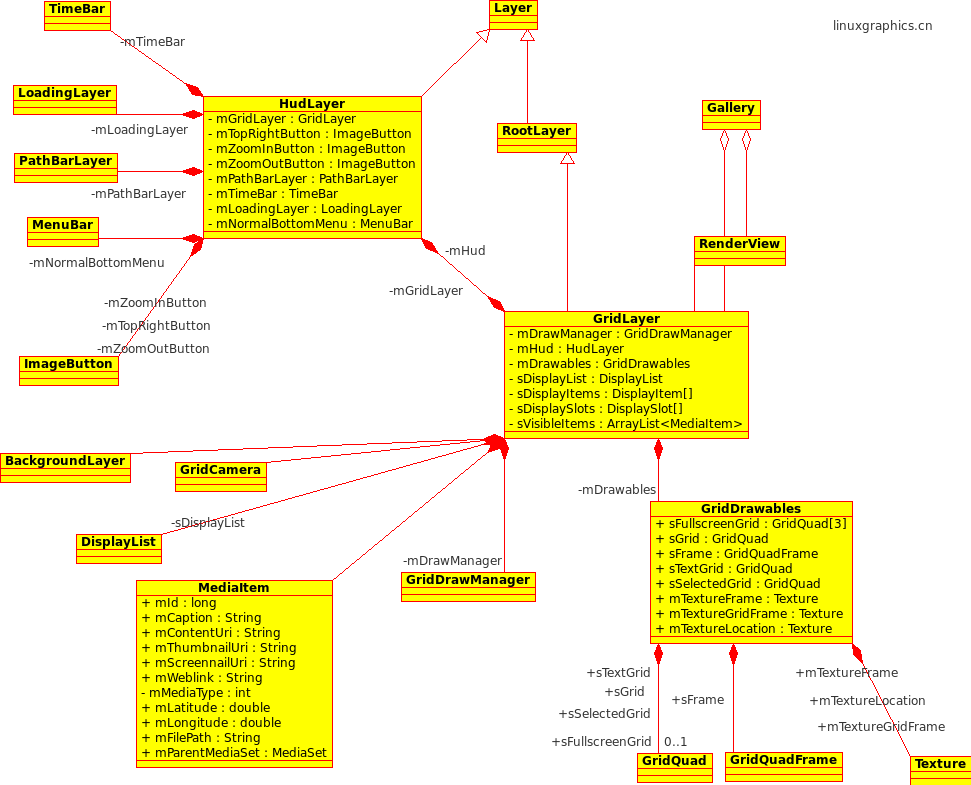
gallery3d 使用了缓冲区对象来保存顶点数据。

参考：

[Nexus One Gallery on Android](http://www.cooliris.com/mobile/nexus-one/)

#### 构件图

gallery3d 的基本构件组成及其关系如下所示：



[**Gallery 3D 代码分析之 GLSurfaceView**](http://yueguc.javaeye.com/blog/749112)

**文章分类:**[**移动开发**](http://www.javaeye.com/blogs/category/mobile)

简介   
  
SDK 中的　android.opengl.GLSurfaceView　类提供如下功能：

* 在 OpenGL ES 和 View 系统之间建立联系；
* 使得 OpenGL ES 可以工作在 Activity 生命周期中；
* 可选择合适的 frame buffer 像素格式；
* 创建并管理一个单独的渲染线程，可以实现平滑的动画；
* 提供　debugging 工具和 API。

一个简单的 GLSurfaceView 应用

**Java代码**

1. **package** com.example.android.apis.graphics;
3. **import** javax.microedition.khronos.egl.EGLConfig;
4. **import** javax.microedition.khronos.opengles.GL10;
6. **import** android.app.Activity;
7. **import** android.opengl.GLSurfaceView;
8. **import** android.os.Bundle;
10. **public** **class** ClearActivity **extends** Activity {
11. @Override
12. **protected** **void** onCreate(Bundle savedInstanceState) {
13. **super**.onCreate(savedInstanceState);
14. mGLView = **new** GLSurfaceView(**this**);
15. mGLView.setRenderer(**new** ClearRenderer());
16. setContentView(mGLView);
17. }
19. @Override
20. **protected** **void** onPause() {
21. **super**.onPause();
22. mGLView.onPause();
23. }
25. @Override
26. **protected** **void** onResume() {
27. **super**.onResume();
28. mGLView.onResume();
29. }
31. **private** GLSurfaceView mGLView;
32. }
34. **class** ClearRenderer **implements** GLSurfaceView.Renderer {
35. **public** **void** onSurfaceCreated(GL10 gl, EGLConfig config) {
36. // Do nothing special.
37. }
39. **public** **void** onSurfaceChanged(GL10 gl, **int** w, **int** h) {
40. gl.glViewport(0, 0, w, h);
41. }
43. **public** **void** onDrawFrame(GL10 gl) {
44. gl.glClear(GL10.GL\_COLOR\_BUFFER\_BIT | GL10.GL\_DEPTH\_BUFFER\_BIT);
45. }
46. }

这个程序功能很简单，每帧绘制时将屏幕设置成黑色。但它是一个完整的工作在 Activity 生命周期中的　OpenGL 程序。当 activity 暂停时，它暂停渲染；当 activity 继续时，它继续渲染。可以将这个程序用作非交互式的 demo 程序。可以在 ClearRenderer.onDrawFrame() 接口中增加 OpenGL 调用做很多的绘制。   
  
GLSurfaceView.Render 接口有三个方法：   
  
    \* onSurfaceCreated():该方法在渲染开始前调用，OpenGL ES 的绘制上下文被重建时也会被调用。当 activity 暂停时绘制上下文会丢失，当 activity 继续时，绘制上下文会被重建。另外，创建长期存在的 OpenGL 资源(如 texture)往往也在这里进行。   
    \* onSurfaceChanged():当 surface 的尺寸发生改变时该方法被调用。往往在这里设置 viewport。若你的 camera 是固定的，也可以在这里设置 camera。   
    \* onDrawFrame():每帧都通过该方法进行绘制。绘制时通常先调用 glClear　函数来清空 framebuffer，然后在调用 OpenGL ES 的起它的接口进行绘制。   
  
输入如何处理   
  
若是开发一个交互型的应用(如游戏)，通常需要子类化 GLSurfaceView，由此可以获取输入事件。下面有个例子：

**Java代码**

1. **package** com.google.android.ClearTest;
3. **import** javax.microedition.khronos.egl.EGLConfig;
4. **import** javax.microedition.khronos.opengles.GL10;
6. **import** android.app.Activity;
7. **import** android.content.Context;
8. **import** android.opengl.GLSurfaceView;
9. **import** android.os.Bundle;
10. **import** android.view.MotionEvent;
12. **public** **class** ClearActivity **extends** Activity {
13. @Override
14. **protected** **void** onCreate(Bundle savedInstanceState) {
15. **super**.onCreate(savedInstanceState);
16. mGLView = **new** ClearGLSurfaceView(**this**);
17. setContentView(mGLView);
18. }
20. @Override
21. **protected** **void** onPause() {
22. **super**.onPause();
23. mGLView.onPause();
24. }
26. @Override
27. **protected** **void** onResume() {
28. **super**.onResume();
29. mGLView.onResume();
30. }
32. **private** GLSurfaceView mGLView;
33. }
35. **class** ClearGLSurfaceView **extends** GLSurfaceView {
36. **public** ClearGLSurfaceView(Context context) {
37. **super**(context);
38. mRenderer = **new** ClearRenderer();
39. setRenderer(mRenderer);
40. }
42. **public** **boolean** onTouchEvent(**final** MotionEvent event) {
43. queueEvent(**new** Runnable(){
44. **public** **void** run() {
45. mRenderer.setColor(event.getX() / getWidth(),
46. event.getY() / getHeight(), 1.0f);
47. }});
48. **return** **true**;
49. }
51. ClearRenderer mRenderer;
52. }
54. **class** ClearRenderer **implements** GLSurfaceView.Renderer {
55. **public** **void** onSurfaceCreated(GL10 gl, EGLConfig config) {
56. // Do nothing special.
57. }
59. **public** **void** onSurfaceChanged(GL10 gl, **int** w, **int** h) {
60. gl.glViewport(0, 0, w, h);
61. }
63. **public** **void** onDrawFrame(GL10 gl) {
64. gl.glClearColor(mRed, mGreen, mBlue, 1.0f);
65. gl.glClear(GL10.GL\_COLOR\_BUFFER\_BIT | GL10.GL\_DEPTH\_BUFFER\_BIT);
66. }
68. **public** **void** setColor(**float** r, **float** g, **float** b) {
69. mRed = r;
70. mGreen = g;
71. mBlue = b;
72. }
74. **private** **float** mRed;
75. **private** **float** mGreen;
76. **private** **float** mBlue;
77. }

其他的 GLSurfaceView 例子   
  
在 Android SDK 的 API Demo例子程序中还有很多例子：

* GLSurfaceView
* Kube
* Translucent GLSurfaceView：透明背景
* Textured Triangle：纹理贴图
* Sprite Text：在 texture 上写文本并显示在 3D 场景中
* Touch Rotate:旋转 3D 对象

选择一个 Surface   
  
GLSurfaceView 提供了接口可选择 surface 的类型。默认情况下， GLSurfaceView 会使用一个 16 位 RGB　frame buffer，带 16 位深度。你也可以根据自己的需要进行选择，比如在　Translucent GLSurfaceView　例子里，需要一个 Alpha 通道来实现透明。GLSurfaceView 提供了 setEGLSurfaceChooser()方法来选择 surface。   
  
选择一个 RGB (565)的16位 framebuffer,接口如下：

**Java代码**

1. setEGLConfigChooser(**boolean** needDepth)

若要定制 red, green, blue, alpha 和 depth，则用如下接口：

**Java代码**

1. setEGLConfigChooser(**int** redSize, **int** greenSize,**int** blueSize, **int** alphaSize,**int** depthSize, **int** stencilSize)

使用自己实现的 EGLConfigChooser，用如下的接口：

**Java代码**

1. setEGLConfigChooser(EGLConfigChooser configChooser)

持续型渲染模式 & 通知型渲染模式   
  
大多数 3D 应用，如游戏、模拟等都是持续型渲染的动画，还有些 3D 应用是反应式的(reactive)，它们往往先被动等待，当用户有了动作再做出反应。对于这种应用，持续渲染屏幕是浪费时间。若开发反应式的应用，可以调用下面的方法

**Java代码**

1. GLSurfaceView.setRenderMode(RENDERMODE\_WHEN\_DIRTY);

停止持续渲染。当调用

**Java代码**

1. GLSurfaceView.requestRender()

时，程序再渲染屏幕。   
调试   
  
GLSurfaceView.setDebugFlags() 方法可以激活 log　或者错误检测，它们可以帮助调试 OpenGL ES 调用。具体使用时，在　GLSurfaceView 的构造函数中，调用 setRender() 之前调用　GLSurfaceView.setDebugFlags()就可以了。下面是个例子：

**Java代码**

1. **public** ClearGLSurfaceView(Context context) {
2. **super**(context);
3. // Turn on error-checking and logging
4. setDebugFlags(DEBUG\_CHECK\_GL\_ERROR | DEBUG\_LOG\_GL\_CALLS);
5. mRenderer = **new** ClearRenderer();
6. setRenderer(mRenderer);
7. }

[**Gallery3d 代码分析之渲染流程**](http://yueguc.javaeye.com/blog/749132)

**文章分类:**[**移动开发**](http://www.javaeye.com/blogs/category/mobile)

RenderView   
  
gallery3d 的渲染从 RenderView 开始。RenderView 从 GLSurfaceView 继承而来，采用了通知型绘制模式，即通过调用 requestRender 通知 RenderView 重绘屏幕。   
  
RenderView 将所有需要绘制的对象都保存一个 Lists中，Lists 包含了5个ArrayList，其定义如下所示：

**Java代码**

1. **public** **final** ArrayList<Layer> updateList = **new** ArrayList<Layer>();
2. **public** **final** ArrayList<Layer> opaqueList = **new** ArrayList<Layer>();
3. **public** **final** ArrayList<Layer> blendedList = **new** ArrayList<Layer>();
4. **public** **final** ArrayList<Layer> hitTestList = **new** ArrayList<Layer>();
5. **public** **final** ArrayList<Layer> systemList = **new** ArrayList<Layer>();

RenderView 的 onDrawFrame 接口完成每一帧的绘制操作，绘制时遍历 lists 里每个 list 的每一个成员并调用其 renderXXX 函数。主要代码如下所示：

**Java代码**

1. ...
2. **final** Lists lists = sLists;
4. **final** ArrayList<Layer> updateList = lists.updateList;
5. **boolean** isDirty = **false**;
6. **for** (**int** i = 0, size = updateList.size(); i != size; ++i) {
7. **boolean** retVal = updateList.get(i).update(**this**, mFrameInterval);
8. isDirty |= retVal;
9. }
10. **if** (isDirty) {
11. requestRender();
12. }
14. // Clear the depth buffer.
15. gl.glClear(GL11.GL\_DEPTH\_BUFFER\_BIT);
16. gl.glEnable(GL11.GL\_SCISSOR\_TEST);
17. gl.glScissor(0, 0, getWidth(), getHeight());
19. // Run the opaque pass.
20. gl.glDisable(GL11.GL\_BLEND);
21. **final** ArrayList<Layer> opaqueList = lists.opaqueList;
22. **for** (**int** i = opaqueList.size() - 1; i >= 0; --i) {
23. **final** Layer layer = opaqueList.get(i);
24. **if** (!layer.mHidden) {
25. layer.renderOpaque(**this**, gl);
26. }
27. }
29. // Run the blended pass.
30. gl.glEnable(GL11.GL\_BLEND);
31. **final** ArrayList<Layer> blendedList = lists.blendedList;
32. **for** (**int** i = 0, size = blendedList.size(); i != size; ++i) {
33. **final** Layer layer = blendedList.get(i);
34. **if** (!layer.mHidden) {
35. layer.renderBlended(**this**, gl);
36. }
37. }
38. gl.glDisable(GL11.GL\_BLEND);

lists 的各个 list 里包含的各个 layer 如下所示：

**Java代码**

1. lists
2. |------------------|-----------------|-----------------|---------------|
3. updateList        opaqueList        blendedList      systemList     hitTestList
4. |                  |                 |                 |            |
5. GridLayer         GridLayer         GridLayer        GridLayer      GridLayer
6. BackgroudLayer    BackgroudLayer    BackgroudLayer
7. HudLayer           HudLayer         HudLayer                        HudLayer
8. TimeBar                             TimeBar                         TimeBar
9. PathBar                             PathBar                         PathBar
10. XXXButton                           XXXButton                       XXXButton
11. XXXMenu                             XXXMenu                         XXXMenu

Layer 类提供了 update(...), renderOpaque (...), renderBlended (...) 接口，从上面 RenderView 的 onDrawFrame 绘制代码可以看到，这些接口被调用。

**Java代码**

1. **public** **abstract** **class** Layer {
2. ... ...
4. **public** **abstract** **void** generate(RenderView view, RenderView.Lists lists);
6. **public** **boolean** update(RenderView view, **float** frameInterval) {
7. **return** **false**;
8. }
10. **public** **void** renderOpaque(RenderView view, GL11 gl) {
11. }
13. **public** **void** renderBlended(RenderView view, GL11 gl) {
14. }
15. ... ...
17. }

GridLayer   
  
GridLayer 中有个 GridDrawManager，专门负责绘制。   
  
下面是 GridDrawManager 的构造函数，从其参数里可以看出些门道。

**Java代码**

1. mDrawManager = **new** GridDrawManager(context, mCamera, mDrawables, sDisplayList, sDisplayItems, sDisplaySlots);

[**Gallery3D笔记**](http://yueguc.javaeye.com/blog/749205) **布局及特效**

**文章分类:**[**移动开发**](http://www.javaeye.com/blogs/category/mobile)

布局及特效  gallery3d 的精华  
一、布局  
gallery3d的界面生成和普通的应用程序不一样。普通程序一般一个界面就是一个activity,布局用xml或代码都可以实现，界面切换是activity的切换方式；而gallery3d没有用android的UI系统，而是用opengl画出来的,即界面是在同一个 activity的，如主界面，缩略图界面，单张图片查看界面，标记界面等都属于同一个activity。那么这界面布局不同的界面是如何组合到一起的呢？分析代码，可以把它看成一个状态机：     
1、标记模式   public static final int MODE\_SELECT = 1;(HudLayer)  
包含了主界面标记模式，缩略界面矩阵游览时标记模式、缩略图界面分类游览时标记模式3个界面  
2、普通模式  public static final int MODE\_NORMAL = 0;(HudLayer)  
包含了

**Java代码**

1. **public** **static** **final** **int** STATE\_MEDIA\_SETS = 0;主界面
2. **public** **static** **final** **int** STATE\_GRID\_VIEW = 1;缩略图矩阵浏览
3. **public** **static** **final** **int** STATE\_FULL\_SCREEN = 2;查看界面
4. **public** **static** **final** **int** STATE\_TIMELINE = 3;缩略图界面分类浏览

有了以上状态分类后，在渲染的时候就能根据些界面的组成来定哪些控件譔隐藏，哪些要显示了。  
下面是基本控件：

**Java代码**

1. com.cooliris.media.GridLayer
2. com.cooliris.media.BackgroundLayer
3. com.cooliris.media.HudLayer
4. com.cooliris.media.ImageButton
5. com.cooliris.media.TimeBar
6. com.cooliris.media.MenuBar
7. com.cooliris.media.PopupMenu
8. com.cooliris.media.PathBarLayer

在渲染时，每一帧所有界面上的元素都画了，由于根据上面的状态只把特定窗口的特定元素显示出来，其它窗口中的隐藏，所以不会乱。  
Layer是上面控件的基类，上面控件的类也就有了下面两个方法来隐藏不譔显示的界面元素。

**Java代码**

1. **public** **boolean** isHidden() {
2. **return** mHidden;
3. }
5. **public** **void** setHidden(**boolean** hidden) {
6. **if** (mHidden != hidden) {
7. mHidden = hidden;
8. onHiddenChanged();
9. }
10. }

下面是根据上面分类来画不同元素所用的标识:

**Java代码**

1. **public** **static** **final** **int** PASS\_THUMBNAIL\_CONTENT = 0;
2. **public** **static** **final** **int** PASS\_FOCUS\_CONTENT = 1;
3. **public** **static** **final** **int** PASS\_FRAME = 2;
4. **public** **static** **final** **int** PASS\_PLACEHOLDER = 3;
5. **public** **static** **final** **int** PASS\_FRAME\_PLACEHOLDER = 4;
6. **public** **static** **final** **int** PASS\_TEXT\_LABEL = 5;
7. **public** **static** **final** **int** PASS\_SELECTION\_LABEL = 6;
8. **public** **static** **final** **int** PASS\_VIDEO\_LABEL = 7;
9. **public** **static** **final** **int** PASS\_LOCATION\_LABEL = 8;
10. **public** **static** **final** **int** PASS\_MEDIASET\_SOURCE\_LABEL = 9;

**Java代码**

1. drawDisplayItem(view, gl, displayItem, texture, PASS\_THUMBNAIL\_CONTENT, placeholder,displayItem.mAnimatedPlaceholderFade); 画缩略图的，注掉此句，前两屏只显示框，第三屏OK
2. drawDisplayItem(view, gl, displayItem, texture, PASS\_FOCUS\_CONTENT, **null**, 0.0f);画单张图片的，注掉，第三屏黑屏
3. drawDisplayItem(view, gl, itemDrawn, textureToUse, PASS\_FRAME, previousTexture, ratio);画边框的，注掉，前两屏明显没有边框，巨齿明显
4. drawDisplayItem(view, gl, displayItem, textureString, PASS\_TEXT\_LABEL, **null**, 0);画文本标签的
5. drawDisplayItem(view, gl, displayItem, textureToUse, PASS\_SELECTION\_LABEL, **null**, 0);画选中标记的
6. drawDisplayItem(view, gl, displayItem, videoTexture, PASS\_VIDEO\_LABEL, **null**, 0);画视频标记的
7. drawDisplayItem(view, gl, displayItem, locationTexture, PASS\_LOCATION\_LABEL, **null**, 0);画位置标记的
8. drawDisplayItem(view, gl, displayItem, locationTexture, PASS\_MEDIASET\_SOURCE\_LABEL,transparentTexture, 0.85f);画源来源图标的(相机或一般文件夹)

二、特效  
举如何显示一张图片为例，在图片完全显示出来经过这样一个过程，附近的图片渐小渐出，当前图片渐大渐入，当前图片逐渐变大直到全屏。实现这个特效，要进行很多帧的渲染。就是说并不是只调一次onDrawFrame函数就可以了，要调用多次。可以把这个特效的实现想成一个状态变化的过程，在每一个状态，纹理的显示大小和位置都不同，这也符合动画的基本原理。放大、缩小我们只要改变顶点数据就可以做到，gallery3d也是这样做的，下面是主要代码：  
我们知道调用onDrawFrame来渲染，最后调到下面的drawFocusItems函数，

**Java代码**

1. GridQuad quad = GridDrawables.sFullscreenGrid[vboIndex];
2. **float** u = texture.getNormalizedWidth();
3. **float** v = texture.getNormalizedHeight();
4. **float** imageWidth = texture.getWidth();
5. **float** imageHeight = texture.getHeight();
6. **boolean** portrait = ((theta / 90) % 2 == 1);
7. **if** (portrait) {
8. viewAspect = 1.0f / viewAspect;
9. }
10. quad.resizeQuad(viewAspect, u, v, imageWidth, imageHeight);//改变用来贴图片的长方形的大小
11. quad.bindArrays(gl);//绑定新数据，为渲染做准备。

而位置的改变有两种方式，一种是直接以顶点数据中改变，另一种是计算出在3维3个方向的偏移量，再调用gltranslate来做，从代码可以看出采用的是第二种方式来做的，比第一种方式更方便一些。代码：

**Java代码**

1. gl.glTranslatef(-translateXf, -translateYf, -translateZf);

而这里的3个偏移量的计算是和camera相关的，相关文件为GridCamera.java，GridCameraManager.java，过程很复杂，理清楚后再细化吧。  
  
cache管理  
下面是cache文件

**Java代码**

1. /sdcard/Android/data/com.cooliris.media/cache/local-album-cache
2. d---rwxr-x system   sdcard\_rw          2010-05-21 09:56 local-album-cache
3. d---rwxr-x system   sdcard\_rw          2010-05-21 09:56 local-meta-cache
4. ----rwxr-x system   sdcard\_rw   299877 2010-05-28 07:36 local-album-cachechunk\_0
5. d---rwxr-x system   sdcard\_rw          2010-05-21 09:56 geocoder-cache
6. ----rwxr-x system   sdcard\_rw      284 2010-05-28 07:36 local-album-cacheindex
7. d---rwxr-x system   sdcard\_rw          2010-05-21 09:56 local-image-thumbs
8. d---rwxr-x system   sdcard\_rw          2010-05-21 09:56 local-video-thumbs
9. d---rwxr-x system   sdcard\_rw          2010-05-21 09:56 picasa-thumbs
10. ----rwxr-x system   sdcard\_rw       80 2010-05-28 07:36 local-meta-cachechunk\_0
11. ----rwxr-x system   sdcard\_rw      164 2010-05-28 07:36 local-meta-cacheindex
12. d---rwxr-x system   sdcard\_rw          2010-05-21 09:56 hires-image-cache
13. ----rwxr-x system   sdcard\_rw   627629 2010-05-28 07:37 local-image-thumbschunk\_0
14. ----rwxr-x system   sdcard\_rw     3914 2010-05-21 09:56 local-image-thumbsindex
15. ----rwxr-x system   sdcard\_rw    53343 2010-05-28 07:34 hires-image-cache-4982941342287215583\_1024.cache
16. ----rwxr-x system   sdcard\_rw   237692 2010-05-28 07:33 hires-image-cache3684568484369117627\_1024.cache
17. ----rwxr-x system   sdcard\_rw   133182 2010-05-28 07:34 hires-image-cache607542544081226432\_1024.cache
18. ----rwxr-x system   sdcard\_rw    83223 2010-05-28 07:34 hires-image-cache4275479623210216146\_1024.cache
19. ----rwxr-x system   sdcard\_rw   292837 2010-05-28 07:34 hires-image-cache-646316556936433937\_1024.cache
20. ----rwxr-x system   sdcard\_rw   191377 2010-05-28 07:35 hires-image-cache2631364604509958174\_1024.cache
21. ----rwxr-x system   sdcard\_rw   366905 2010-05-28 07:35 hires-image-cache-3280562009766080884\_1024.cache
22. ----rwxr-x system   sdcard\_rw   323671 2010-05-28 07:35 hires-image-cache5752471827533329222\_1024.cache

创建cache的关键代码

**Java代码**

1. LocalDataSource
2. **public** **static** **final** DiskCache sThumbnailCache = **new** DiskCache("local-image-thumbs");----------------------local-image-thumbs local-image-thumbschunk\_0  local-image-thumbsindex
3. **public** **static** **final** DiskCache sThumbnailCacheVideo = **new** DiskCache("local-video-thumbs");--------------------local-video-thumbs
4. **public** **static** **final** DiskCache sAlbumCache = **new** DiskCache("local-album-cache");----------------------local-album-cache  local-album-cacheindex
5. **public** **static** **final** DiskCache sMetaAlbumCache = **new** DiskCache("local-meta-cache");------------------local-meta-cache  local-meta-cacheindex
6. getChunkFile --------------local-meta-cachechunk\_0  local-album-cachechunk\_0
8. ReverseGeocoder::  **private** **static** **final** DiskCache sGeoCache = **new** DiskCache("geocoder-cache"); -------------------------geocoder-cache
9. PicasaDataSource:: **public** **static** **final** DiskCache sThumbnailCache = **new** DiskCache("picasa-thumbs");-----------------------------picasa-thumbs
10. UriTexture::writeToCache  --------------------------hires-image-cache-xxx\_1024.cache

布局补充:  
在画一个界面是,是分类化的,比如第一个界面是显示所有有图片的文件夹,在代码里叫专辑.有这些元素要创建:  
文本标签  显示专辑名和专辑内图片或视频数  
路径条 显示路径名  
按纽 拍照按纽,放大/缩小  
菜单栏 全选,取消全选,分享,删除,更多等  
图片边框  
用于显示图片的矩形  
在渲染时一次把一类元素画完,再画另一类.如主界面顺序为:  
路径条->按纽->文本标签->图片边框->图片.  
具体代码见drawBlendedComponents函数

1.CacheService.java 中写缓存： sAlbumCache.put(ALBUM\_CACHE\_LOCALE\_INDEX, sDummyData, 0);

  第一个是 key ，这里是正常数据，当然还有别的 key ， key 分别是 -1,-2,-3,-4,-5 。

2.DiskCache.java 中，执行上面的写的过程，这里先得明白他的 cache 怎么装的：

    它是由很多称之为“片”的文件组成的，形成一个 List 形式： private final

LongSparseArray<RandomAccessFile> mChunkFiles = new LongSparseArray<RandomAccessFile>();

即 mChuckFiles 就是整个 cache ，里面包括很多 chunk( 即片 ) ，每一个 chunk 大小为 1MB.

当要写入某一个 chunk 里面的时候，先要找到他在 mChuckFiles 里面的索引值即 chunkIndex, 由

mChunkFiles.get(chunkIndex); 来获取这个文件， chunkIndex 怎么来的呢？

private LongSparseArray<Record> mIndexMap;

Record record = mIndexMap.get(key); 这里的 key 就是上面用 put 方法传过来的

ALBUM\_CACHE\_LOCALE\_INDEX 的值（就是 -5 ）

int chunkIndex = record.chunk;

这么一步步来的。

当然了，第一次都是空的，也就是 get 不到东西 mChunkFiles.get(chunkIndex); 和 Record record =

mIndexMap.get(key); 都 get 不到，那么第一次就先把东西放进去， mIndexMap.put(key, new Record

(chunkIndex, record.offset, data.length, record.sizeOnDisk, timestamp)); （记录 key 值）以及 final

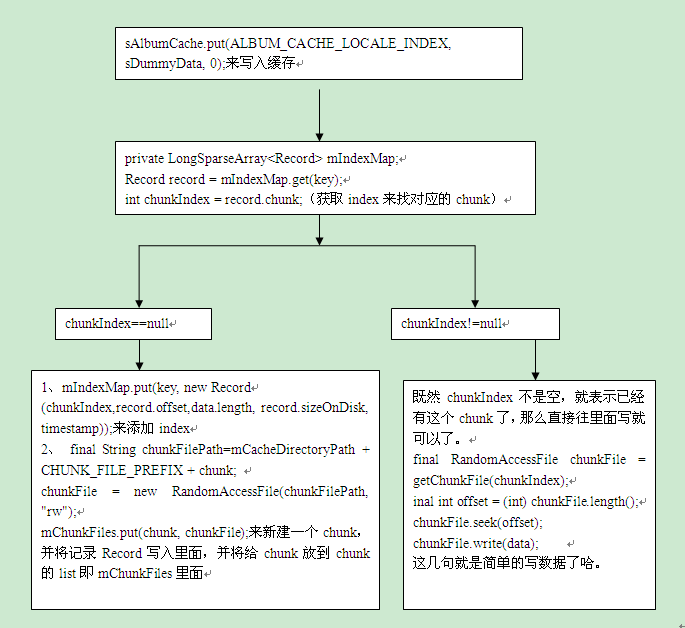
String chunkFilePath = mCacheDirectoryPath + CHUNK\_FILE\_PREFIX + chunk; chunkFile = new

RandomAccessFile(chunkFilePath, "rw");mChunkFiles.put(chunk, chunkFile); （三句代码来新建一个

chunkfile 并放到 cache 列表里面）

注意： Record 是内部类，只是一个数据集合类而已，相当于文件描述信息。每个 cache （即 chunk ）对应一个。

private final LongSparseArray<RandomAccessFile> mChunkFiles = new LongSparseArray<RandomAccessFile>(); 中mChunkFiles 最大装 13 个，每个 chunk 是 1M, 所以全部 Cache 是 13M.



[**Gallery3D中画图时调用glTranslate函数参数赋值过程**](http://yueguc.javaeye.com/blog/749233)

**文章分类:**[**移动开发**](http://www.javaeye.com/blogs/category/mobile)

GridDrawManager::drawDisplayItem(RenderView view, GL11 gl, DisplayItem displayItem, Texture texture, int pass,Texture previousTexture, float mixRatio) 函数有下面几句： 

**Java代码**

1. Vector3f animatedPosition = displayItem.mAnimatedPosition;
2. **float** translateXf = animatedPosition.x \* camera.mOneByScale;
3. **float** translateYf = animatedPosition.y \* camera.mOneByScale;
4. **float** translateZf = -animatedPosition.z;

调用过程：   
->computeVisibleItems(),displayItems[baseIndex + j] = displayItem;Vector3f position = pool.create(); GridCameraManager.getSlotPositionForSlotIndex(i, camera, layout, deltaAnchorPosition, position);//give position value  displayList.setPositionAndStackIndex(displayItem, position, j, true);//raletive position to item   
  
->GridLayer, mDrawManager = new GridDrawManager(context, mCamera, mDrawables, sDisplayList, sDisplayItems, sDisplaySlots);   
->GridDrawManager(), mDisplayItems = displayItems;   
->drawFocusItems ,DisplayItem[] displayItems = mDisplayItems;   
->animatedPosition = displayItem.mAnimatedPosition;   
->drawDisplayItem, amAnimatedPosition   
->DisplayItem::commit()  amAnimatedPosition.set(mTargetPosition);   
-> DisplayItem::set(Vector3f position, int stackIndex, boolean performTransition) mTargetPosition.z = 

**Java代码**

1. **public** **void** getPositionForSlotIndex(**int** slotIndex, **int** itemWidth, **int** itemHeight, Vector3f outPosition) {
2. outPosition.x = (slotIndex / mNumRows) \* (itemWidth + mSpacingX);
3. outPosition.y = (slotIndex % mNumRows) \* (itemHeight + mSpacingY);
4. **int** maxY = (mNumRows - 1) \* (itemHeight + mSpacingY);
5. outPosition.y -= (maxY >> 1);
6. outPosition.z = 0;
7. Log.d("outPosition","slotIndex="+slotIndex+",mNumRows="+mNumRows+",outPosition=("+outPosition.x+","+outPosition.y+","+outPosition.z+")");
8. }

在gallery3d中矩阵是从上到下、从左到右排列的，在主界面时最多有3行，mNumRows＝3，在缩略图界面最多4行mNumRows＝4，在查看界面只有一行mNumRows＝1   
上面函数是计算所绘制项位置的，slotIndex / mNumRows得到的是当前处于多少列，slotIndex % mNumRows得到的是处于多少行。 int maxY = (mNumRows - 1) \* (itemHeight + mSpacingY);   
        outPosition.y -= (maxY >> 1);是为了在y方向对称，想成屏幕中间是分界线，以上的项y为负，线以下的y为正。   
deltaAnchorPosition的赋值过程：

**Java代码**

1. **int** currentlyVisibleSlotIndex = getAnchorSlotIndex(ANCHOR\_CENTER);->anchorItem = displayItem.mItemRef;-> newSlotIndex = i;->**if** (currentAnchorSlotIndex != Shared.INVALID && newAnchorSlotIndex != Shared.INVALID) {
2. layout.getPositionForSlotIndex(newAnchorSlotIndex, itemWidth, itemHeight, deltaAnchorPosition);
3. oldLayout.getPositionForSlotIndex(currentAnchorSlotIndex, itemWidth, itemHeight, currentSlotPosition);
4. currentSlotPosition.subtract(sDeltaAnchorPosition);
5. deltaAnchorPosition.subtract(currentSlotPosition);
6. deltaAnchorPosition.y = 0;
7. deltaAnchorPosition.z = 0;
8. }

MediaFeed::run()   
onFeedChanged(, onLayout(newSlotIndex, currentlyVisibleSlotIndex, null);   
onLayout,  sDeltaAnchorPositionUncommited.set(deltaAnchorPosition);   
computeVisibleRange(),sDeltaAnchorPosition.set(sDeltaAnchorPositionUncommited);   
deltaAnchorPosition.set(sDeltaAnchorPosition);   
deltaAnchorPosition

### [Gallery3D各个界面可见范围计算方法](http://yueguc.javaeye.com/blog/749234)

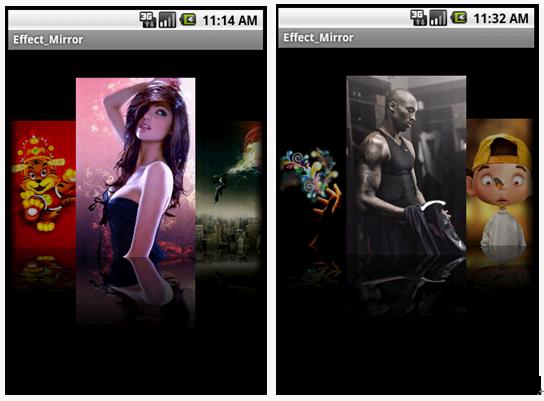
**文章分类:**[**移动开发**](http://www.javaeye.com/blogs/category/mobile)

computeVisibleRange算法分析：   
第1步，计算出left,right,bottom,top   
第2步，计算出numSlots，并除于2赋值给index   
第3步，由index得position,判断position是否在第1步计算出的范围内，是的话，就把第2步计算得出的中间的index赋值给 firstVisibleSlotIndex，lastVisibleSlotIndex，否则，根据滑动窗口算法改变index直到求组所需index   
第4步，在while循环中，用第3步得到的firstVisibleSlotIndex求出position,进行和第2步相反的判断，即 position若不在可视范围内，则将相应的index给firstVisibleSlotIndex，否则减 firstVisibleSlotIndex,直到找到最小的可视范围内的index作为firstVisibleSlotIndex。   
第5步，在while循环中，用第3步得到的lastVisibleSlotIndex求出position,进行和第2步相反的判断，即 position若不在可视范围内，则将相应的index给lastVisibleSlotIndex，否则增 lastVisibleSlotIndex,直到找到可视范围内的最大的index作为lastVisibleSlotIndex。   
第6步，进行firstVisibleSlotIndex，lastVisibleSlotIndex的越界判断。 outBufferedVisibleRange对应的是可见的。outBufferedVisibleRange对应的是0～文件夹的最大数。   
computeVisibleItems算法分析：   
第1步 由slot计算出position,set,当前set不为空且slot在有效范围，创建bestItems，计算sortedIntersection   
第2步 计算这个slotindex中的图片数目，取这个文件中的前12张图片加到bestItems.   
第3步  取bestItems里的图片对应的displayList中的displayItem，并赋值给displayItems数组，同时保存 position,及j，j是bestItems数组中一项，范围是0～12。   
第四步  对于每一个文件夹，要在displayItems里有对应的12项，当文件夹内图片不足12时，余下的用null填充。   
当绘制缩略图界面时，有些不同   
在第1步中，slotindex不再表示文件夹，这时表示具体某一张图片了，所以由slot得到的set里始终只有1项，且会调 ArrayUtils.computeSortedIntersection(visibleItems, items, MAX\_ITEMS\_PER\_SLOT, bestItems, sTempHash);给bestItems赋值，这样第2步就在bestItems加项动作不执行。

### [镜像倒影特效Gallery](http://yueguc.javaeye.com/blog/750986)

**文章分类:**[**移动开发**](http://www.javaeye.com/blogs/category/mobile)

#### 效果展示

[](http://icekirin.com/wordpress/wp-content/uploads/2010/03/18.jpg)

本文档将介绍在android上如何实现一个倒影效果的Gallery。

为了达到上图所示的效果，

首先，是需要有一个自定义的gallery，实现了对Gallery的继承，通过重写getChildStaticTransformation方 法来控制，每个子view有不同的缩放比例，形成阶梯状的展示。这个gallery是在坤庭的代码基础上实现的，我这里就不再重复介绍。

接下来，gallery中的每个view，都是一个自定义的MirrorView，由它来实现了画原图及画倒影。新的算法解决了性能问题，下面就重 点说一下这部分内容:

镜像特效最近还蛮流行的，尤其在HTC 的Sense 介面上，常常都可以见到。大家可以看到，加了个镜像特效后，那感觉就很立体，感觉上好像是这些图片摆立在一个玻璃桌面上。

在Android 上要帮图片加上镜像特效，会不会很麻烦？一点也不麻烦，只要几行代码，就可以搞定。

因此，在开始看答案之前，我会建议你要先有Photoshop 的使用经验。想想，如果用Photoshop 要帮图片加上镜像特效，要如何做？我想一般不外乎是先复制个图片，并将其垂直翻转，接着再对这翻转的图片，加个由灰到黑的渐层mask 即可。

好了，让我们来看一下答案。底下就是帮图片加上镜像特效的程式范例。

**Java代码**

1. **public**  **class**  MirrorView **extends**  View {
3. Paint m\_paint;
5. **int** m\_nShadowH;
7. Drawable m\_dw;
9. Bitmap m\_bitmap;
11. Matrix mMatrix;
13. **int** shadowHeight;
15. **public** MirrorView(Context context, Bitmap bitmap) {
17. **super** (context);
19. m\_bitmap = bitmap;
21. \_Init();
23. }
25. **private** **void** \_Init() {
27. //m\_dw = new BitmapDrawable(BitmapFactory.decodeResource(getResources(),  R.drawable.icon));
29. m\_dw = **new** BitmapDrawable(m\_bitmap);
31. m\_dw.setBounds(0,0,m\_dw.getIntrinsicWidth(),m\_dw.getIntrinsicHeight());
33. m\_nShadowH = m\_dw.getIntrinsicHeight()/1;
35. m\_paint = **new** Paint(Paint.ANTI\_ALIAS\_FLAG );
37. LinearGradient lg = **new** LinearGradient(0, 0, 0, m\_nShadowH, 0xB0FFFFFF, 0×00000000, Shader.TileMode.CLAMP );
39. m\_paint.setShader(lg);
41. m\_paint.setXfermode(**new** PorterDuffXfermode(PorterDuff.Mode.MULTIPLY ));
43. mMatrix = **new** Matrix();
45. }
47. @Override
49. **public** **void** onDraw(Canvas canvas) {
51. **super** .onDraw(canvas);
53. **int** nX = 0;
55. **int** nY = 0;
57. \_DrawNormalImg(canvas, nX, nY);
59. \_DrawMirror(canvas, nX, nY);
61. }
63. **private** **void** \_DrawNormalImg(Canvas canvas, **int** nX, **int** nY) {
65. canvas.save(Canvas.MATRIX\_SAVE\_FLAG );
67. canvas.translate(nX, nY);
69. m\_dw.draw(canvas);
71. canvas.restore();
73. }
75. **private** **void** \_DrawMirror(Canvas canvas, **int** nX, **int** nY) {
77. **int** nW = m\_dw.getIntrinsicWidth();
79. **int** nH = m\_dw.getIntrinsicHeight();
81. shadowHeight=nH/2;
83. **float** [] src={0, nH, nW, nH, nW,nH – m\_nShadowH, 0, nH – m\_nShadowH};
85. **float** [] dst={ 0, nH, nW, nH,nW, shadowHeight, 0, shadowHeight };
87. canvas.save();
89. mMatrix.setPolyToPoly(src, 0, dst, 0, src.length >> 1);
91. canvas.concat(mMatrix);
93. //draw mirror image
95. canvas.save(Canvas.MATRIX\_SAVE\_FLAG );
97. canvas.scale(1.0f, -1.0f);
99. canvas.translate(nX, -(nY + nH \* 2));
101. canvas.clipRect(0, nH, nW, nH – m\_nShadowH);
103. m\_dw.draw(canvas);
105. canvas.restore();
107. //draw mask
109. canvas.save();
111. canvas.translate(nX, nY + nH);
113. canvas.drawRect(0, 0, nW, m\_nShadowH, m\_paint);
115. canvas.restore();
117. canvas.restore();
119. }
121. }

\_DrawMirror() 方法是关键。用Photoshop 要如何做出镜像特效？第一步是先画出垂直翻转的图片。

Android 绘图座标体系预设的原点在左上角，X 轴往右是越来越大的正值，而Y 轴往下，则是越来越大的正值。要画出垂直翻转的图片，其实也就是要垂直翻转整个绘图座标体系。在 Android 中，要如何做？答案就是 canvas.scale(1.0f, -1.0f)。很简单吧，没想到给scale() 函式一个负值，就可以翻转相对应的轴。

在Photoshop 中，做镜像特效的第二步是要对这翻转的图片，加个由灰到黑的渐层mask。

在Android 中，要画渐层色，那就一定得用LinearGradient 这个类别。至于要对背景图加上个mask，就请参考一下Paint 的setXfermode() 函式。 \_Init() 这个函式，就是负责生成一个由灰到黑渐层mask 的m\_paint 物件。

这个控件我测试过，200张图片加入adapter，在大数据量情况下性能也没有问题。

[**菱形3D实例**](http://yueguc.javaeye.com/blog/751561)

**文章分类:**[**移动开发**](http://www.javaeye.com/blogs/category/mobile)

下面是具体的实现方法：   
首先需要建两个array，第一array是用来告诉opengl这个图形有哪些顶点：   
画一个三维的坐标轴，然后把你要画的点都算出来，然后放在这个array里。

**Java代码**

1. **float** l=1.5f;
2. **float**[] vertex={
4. 0.0f,l,0.0f,
6. l,0.0f,0.0f,
8. 0.0f,0.0f,l,
10. -l,0.0f,0.0f,
12. 0.0f,0.0f,-l,
14. 0.0f,-l,0.0f
16. };

第二个array是告诉opengl 你要怎样组织这些点：   
这里我要画三角形，所以每三个点是一组。

**Java代码**

1. **byte**[] edge=
2. {
3. 0,1,2,
4. 1,2,5,
6. 0,2,3,
7. 5,2,3,
9. 0,3,4,
10. 5,3,4,
12. 0,4,1,
13. 5,4,1
15. };

这里的数字，是第一个array的index。   
下面你要建立两个Buffer它们是用来存放这两个array的。

**Java代码**

1. ByteBuffer bb = ByteBuffer.allocateDirect(vertex.length\*4);
2. bb.order(ByteOrder.nativeOrder());
3. fbv=bb.asFloatBuffer();
4. fbv.put(vertex);
5. fbv.position(0);
7. ffe=ByteBuffer.allocateDirect(edge.length);
8. ffe.put(edge);
9. ffe.position(0);

这样一个三维的菱形就画好了。   
  
下面你要写一个方法能让它自己把自己画出来！

**Java代码**

1. **public** **void** draw(GL10 gl)
2. {
3. gl.glFrontFace(GL10.GL\_CW);
4. gl.glVertexPointer(3, GL10.GL\_FLOAT, 0, fbv);
5. gl.glDrawElements(GL10.GL\_TRIANGLES, 24, GL10.GL\_UNSIGNED\_BYTE, ffe);
7. }

先说第一个glFrontFace，物体都有一个正面一个反面，这里告诉opengl显示这个物体按顺时针方向（CW=> clockwise）   
  
gl.glVertexPointer(3, GL10.GL\_FLOAT, 0, fbv);这个方法是把本程序所用的点都传递个opengl。opengl需要知道什么哪？首先是 这个点是几维的（opengl 支持2，3，4 维），这里是3所以是三维的，第二个参数告诉opengl，这个点是用什么样类型的变量来储存的，这里是float类型。第三个是步长（stride），这个我还没弄明白，不过我看的例子都为0. 最后把你建立好的三维坐标点都传给opengl   
  
gl.glDrawElements。 这个方法是告诉opengl如果要画这个图形，应该怎么画。第一个参数，告诉opengl 用画三角形（这样opengl就以三个点为一组），然后告诉opengl你要用到多少个点（注意这个点是在第二个array里的点数）。 第三个是告诉opengl这些点（其实是三维坐标点的reference）的类型。这里是unsigned byte。最后把你排列点的array 放进去！   
  
第二个大的步骤是创建一个让这个三维坐标运行的环境（Renderer）。   
这是一个interface类   
首先，在onDrawFrame里，我们告诉本程序这个三维图形的行为：   
在做任何事情之前，我们要清空所有以前内存里的东西，这个内存包括：Color 和Depth   
gl.glClear(GL10.GL\_COLOR\_BUFFER\_BIT|GL10.GL\_DEPTH\_BUFFER\_BIT);   
然后告诉opengl你要用那个MatrixMode 这个就比较难解释了。   
如果写程序 只要记住 GL\_MODELVIEW 是管理图形的 缩放，移动，和转动就行了.(如果那个朋友想理解的更深一点，可以联系我，我可以把我的笔记发过去或者参考 http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/projection/projection\_viewing.html )。   
  
  
gl.glTranslatef(0, 0, -3.0f);   
这个方法告诉opengl把图形沿z轴迁移3个unit。这三个值分别是x，y，z轴。   
gl.glRotatef(angle,0, 1, 0);   
这个方法告诉我们以y为轴。 转angle个度数。注意这里的1和0是告诉沿着那个轴转，别的值应该没有意义。   
gl.glEnableClientState(GL10.GL\_VERTEX\_ARRAY);   
还记得上边说的opengl分client side和service side吗。 这个是告诉opengl如果client side调用draw什么的时候，这个vertex array是可用的。opengl有很多这样的可选项，所以需要告诉opengl，因为我们已经设置了vertex array（我们的第一个array）,所以告诉opengl 它是可用的（如果不告诉，opengl会忽略）！   
  
trian.draw(gl);   
这个方法是把图形画出来。   
  
angle++;   
为了达到动态的效果，我们让每一个frame 的angle，比上一个多一度。   
当显示空间大小发生变化的时候，我们应该告诉opengl一下信息：

**Java代码**

1. **public** **void** onSurfaceChanged(GL10 gl, **int** width, **int** height)
2. {
3. gl.glViewport(0, 0, width, height);
4. gl.glMatrixMode(GL10.GL\_PROJECTION);
5. gl.glLoadIdentity();
6. **float** ratio = (**float**)width/height;
7. gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
9. }

首先是opengl可用的空间 ：   
gl.glViewport(0, 0, width, height);   
想象一下用这四个点画出来的四边形，就是opengl所能用的空间。   
  
gl.glMatrixMode(GL10.GL\_PROJECTION);   
这个matrix是如何把三维坐标转换为二维坐标并把它放在显示器上。   
gl.glLoadIdentity   
是告诉opengl初始化这个matrix。   
  
gl.glFrustumf   
要把三维的东西用二维显示出来，需要知道几个东西，第一是这个显示平面有多大，你可以看多近和多远。这里的头四个参数，建立了一个四边形，告诉opengl把图形显示在这个范围了。后两个参数告诉opengl这里显示平面里可以显示三维空间里最近和最远的位置。   
  
当这个三维图形建立的是时候，我们可以告诉opengl一些基本参数。这里把能省略的都省略了（其实什么都没有也可以运行！）

**Java代码**

1. **public** **void** onSurfaceCreated(GL10 gl, EGLConfig arg1)
2. {
3. gl.glEnable(GL10.GL\_DEPTH\_TEST);
4. gl.glClearColor(0,0, 0, 0);
5. }

gl.glEnable(GL10.GL\_DEPTH\_TEST); 告诉opengl要检查depth，为什么哪。在三维空间里一个物体A在另一个物体B后面，那么这个A被B挡住里，所以你是看不见的。我们要告诉opengl，我们不想看见被挡住的东西。这个GL\_DEPTH\_TEST 就是这个功能。   
gl.glClearColor(0,0, 0, 0);   
设置这个背景颜色为黑色，应为我们没有给我们的三维图形设置颜色（为了简单），它的初始化颜色是白色。   
  
以下是源代码：

**Java代码**

1. **package** Beta.ThreeD;
3. **import** java.nio.ByteBuffer;
4. **import** java.nio.ByteOrder;
5. **import** java.nio.FloatBuffer;
7. **import** javax.microedition.khronos.opengles.GL10;
9. **public** **class** TriangleShape
10. {
12. **private** **final** **float** l=1.5f;
14. **private** FloatBuffer fbv;
16. **private** ByteBuffer ffe;
18. **public** TriangleShape()
20. {
22. **float**[] vertex={
24. 0.0f,l,0.0f,
26. l,0.0f,0.0f,
28. 0.0f,0.0f,l,
30. -l,0.0f,0.0f,
32. 0.0f,0.0f,-l,
34. 0.0f,-l,0.0f
36. };

39. **byte**[] edge=
41. {
43. 0,1,2,
45. 1,2,5,


49. 0,2,3,
51. 5,2,3,


55. 0,3,4,
57. 5,3,4,


61. 0,4,1,
63. 5,4,1
65. };
67. ByteBuffer bb = ByteBuffer.allocateDirect(vertex.length\*4);
69. bb.order(ByteOrder.nativeOrder());
71. fbv=bb.asFloatBuffer();
73. fbv.put(vertex);
75. fbv.position(0);



80. ffe=ByteBuffer.allocateDirect(edge.length);
82. ffe.put(edge);
84. ffe.position(0);
86. }
88. **public** **void** draw(GL10 gl)
90. {
92. gl.glFrontFace(GL10.GL\_CW);
94. gl.glVertexPointer(3, GL10.GL\_FLOAT, 0, fbv);
96. gl.glDrawElements(GL10.GL\_TRIANGLES, 24, GL10.GL\_UNSIGNED\_BYTE, ffe);
98. }
99. }

102. **package** Beta.ThreeD;
104. **import** javax.microedition.khronos.egl.EGLConfig;
105. **import** javax.microedition.khronos.opengles.GL10;
107. **import** android.opengl.GLSurfaceView.Renderer;
109. **public** **class** MySimpleRendered **implements** Renderer
110. {
112. **private** **int** angle=50;
114. **private** TriangleShape trian;
116. **public** MySimpleRendered()
118. {
120. trian = **new** TriangleShape();
122. }
124. @Override
126. **public** **void** onDrawFrame(GL10 gl)
128. {
130. // TODO Auto-generated method stub
132. gl.glClear(GL10.GL\_COLOR\_BUFFER\_BIT|GL10.GL\_DEPTH\_BUFFER\_BIT);
134. gl.glMatrixMode(GL10.GL\_MODELVIEW);
136. gl.glLoadIdentity();
138. gl.glTranslatef(0, 0, -3.0f);
140. gl.glRotatef(angle,0, 1, 0);
142. gl.glRotatef(angle, 1, 0, 0);
144. gl.glEnableClientState(GL10.GL\_VERTEX\_ARRAY);
146. trian.draw(gl);
148. angle++;
150. }

153. @Override
155. **public** **void** onSurfaceChanged(GL10 gl, **int** width, **int** height)
157. {
159. // TODO Auto-generated method stub
161. gl.glViewport(0, 0, width, height);
163. gl.glMatrixMode(GL10.GL\_PROJECTION);
165. gl.glLoadIdentity();
167. **float** ratio = (**float**)width/height;
169. gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);


173. }

176. @Override
178. **public** **void** onSurfaceCreated(GL10 gl, EGLConfig arg1)
180. {
182. gl.glEnable(GL10.GL\_DEPTH\_TEST);
184. gl.glClearColor(0,0, 0, 0);
186. }
188. }
189. **package** Beta.ThreeD;
191. **import** android.app.Activity;
192. **import** android.os.Bundle;
193. **import** android.opengl.GLSurfaceView;
194. **public** **class** Triangle **extends** Activity {
195. /\*\* Called when the activity is first created. \*/
197. **private** GLSurfaceView my\_view;
198. @Override
199. **public** **void** onCreate(Bundle savedInstanceState) {
200. **super**.onCreate(savedInstanceState);
201. setContentView(R.layout.main);
202. my\_view = **new** GLSurfaceView(**this**);
203. my\_view.setRenderer(**new** MySimpleRendered());
204. **this**.setContentView(my\_view);
205. }
206. **public** **void** onResume()
207. {
209. **super**.onResume();
211. my\_view.onResume();
212. }
213. **public** **void** onPause()
214. {
216. **super**.onPause();
218. my\_view.onPause();
219. }
220. }

[**3D桌面效果动画类**](http://yueguc.javaeye.com/blog/751576)

**文章分类:**[**移动开发**](http://www.javaeye.com/blogs/category/mobile)

**Java代码**

1. **public** **class** CubeAnimation **extends** Animation **implements**
2. Animation.AnimationListener {
3. **public** **static** **final** **int** FACE\_LEFT = 0;
4. **public** **static** **final** **int** FACE\_RIGHT = 1;
5. **private** **int** mInd;
6. **private** **float** mFromDegrees;
7. **private** **float** mToDegrees;
8. **private** **float** mCenterX;
9. **private** **float** mCenterY;
10. **private** **int** mHorizonType = 1;
11. **private** **float** mHorizonValue = 0.5F;
12. **private** Camera mCamera;
13. **private** View mView;
15. **public** CubeAnimation(**int** ind) {
16. **this**.mInd = ind;
17. **this**.mFromDegrees = 0.0F;
18. **this**.mToDegrees = 90.0F;
19. }
21. **public** CubeAnimation(Context context, AttributeSet attrs) {
22. **super**(context, attrs);
23. TypedArray a = context.obtainStyledAttributes(attrs,
24. R.styleable.CubeAnimation);
26. **this**.mFromDegrees = 0.0F;
27. **this**.mToDegrees = 90.0F;
29. Description d = Description.parseValue(a.peekValue(0));
30. **this**.mHorizonType = d.type;
31. **this**.mHorizonValue = d.value;
33. **this**.mInd = a.getInt(1, 0);
35. **boolean** t = a.getBoolean(2, **true**);
36. **if** (!(t)) {
37. **this**.mInd = (1 - **this**.mInd);
38. **this**.mToDegrees = 0.0F;
39. **this**.mFromDegrees = 90.0F;
40. }
41. a.recycle();
42. }
44. **public** **void** onAnimationStart(Animation anim) {
45. **this**.mView.setVisibility(View.VISIBLE);
46. }
48. **public** **void** onAnimationEnd(Animation anim) {
49. **this**.mView.setVisibility((**this**.mInd == 0) ? 0 : 8);
50. **this**.mInd = (1 - **this**.mInd);
51. }
53. **public** **void** onAnimationRepeat(Animation anim) {
54. }
56. **public** **static** **void** startCubeAnimation(Context context, **int** id, View view1,
57. View view2) {
58. XmlPullParser parser;
59. **try** {
60. parser = context.getResources().getAnimation(id);
61. AttributeSet attrs = Xml.asAttributeSet(parser);
63. **int** type = parser.getEventType();
64. **int** depth = parser.getDepth();
65. **while** (**true**) {
66. **while** (**true**) {
67. **if** ((((type = parser.next()) == 3) && (parser.getDepth() <= depth))
68. || (type == 1))
69. **break** label172;
70. **if** (type == 2)
71. **break**;
72. }
74. String name = parser.getName();
76. **if** (name.equals("cube")) {
77. CubeAnimation anim1 = **new** CubeAnimation(context, attrs);
78. anim1.mInd = 1;
79. anim1.mView = view1;
80. anim1.setAnimationListener(anim1);
81. CubeAnimation anim2 = **new** CubeAnimation(context, attrs);
82. anim2.mInd = 0;
83. anim2.mView = view2;
84. anim2.setAnimationListener(anim2);
85. view1.startAnimation(anim1);
86. label172: view2.startAnimation(anim2);
87. }
88. }
89. } **catch** (Resources.NotFoundException ex) {
90. Log.e("CubeAnimation", "NotFoundException");
91. } **catch** (XmlPullParserException ex) {
92. Log.e("CubeAnimation", "XmlPullParserException");
93. } **catch** (IOException ex) {
94. Log.e("CubeAnimation", "IOException");
95. }
96. }
98. **public** **void** initialize(**int** width, **int** height, **int** parentWidth,
99. **int** parentHeight) {
100. **super**.initialize(width, height, parentWidth, parentHeight);
101. **this**.mCenterX = resolveSize(1, 0.5F, width, parentWidth);
102. **this**.mCenterY = resolveSize(1, 0.5F, height, parentHeight);
103. **if** (**this**.mHorizonType == 0) {
104. **this**.mHorizonValue /= height;
105. }
107. **this**.mCamera = **new** Camera();
108. }
110. **protected** **void** applyTransformation(**float** interpolatedTime, Transformation t) {
111. **float** fromDegrees = **this**.mFromDegrees;
112. **float** degrees = fromDegrees + (**this**.mToDegrees - fromDegrees)
113. \* interpolatedTime;
115. **float** centerX = **this**.mCenterX;
116. **float** centerY = **this**.mCenterY;
117. Camera camera = **this**.mCamera;
119. Matrix matrix = t.getMatrix();
121. camera.save();
123. **float** b = 0.0F;
124. **float** e = -**this**.mHorizonValue;
126. **if** (**this**.mInd == 0) {
127. degrees += 90.0F;
128. }
130. camera.rotateY(degrees);
132. camera.getMatrix(matrix);
133. camera.restore();
135. **if** (**this**.mInd == 0) {
136. matrix.preScale(-1.0F, 1.0F, centerX, 0.0F);
137. }
139. **float** tranX = 320.0F \* interpolatedTime;
141. **float** tranY = -centerY \* e + b;
142. matrix.preTranslate(0.0F, centerY \* e);
143. matrix.postTranslate(tranX, tranY);
144. }
146. **protected** **static** **class** Description {
147. **public** **int** type;
148. **public** **float** value;
150. **static** Description parseValue(TypedValue value) {
151. Description d = **new** Description();
152. **if** (value == **null**) {
153. d.type = 0;
154. d.value = 0.0F;
155. } **else** {
156. **if** (value.type == 6) {
157. d.type = (((value.data & 0xF) == 1) ? 2 : 1);
159. d.value = TypedValue.complexToFloat(value.data);
160. **return** d;
161. }
162. **if** (value.type == 4) {
163. d.type = 0;
164. d.value = value.getFloat();
165. **return** d;
166. }
167. **if** ((value.type >= 16) && (value.type <= 31)) {
168. d.type = 0;
169. d.value = value.data;
170. **return** d;
171. }
172. }
174. d.type = 0;
175. d.value = 0.0F;
177. **return** d;
178. }
179. }
180. }

[**gallery3d源码学习总结（一）——绘制流程drawFocusItems**](http://www.eoeandroid.com/thread-41048-1-1.html)

**显示单张图片相关的输入变量**

1. int selectedSlotIndex = mSelectedSlot;
2. GridDrawables drawables = mDrawables;
3. GridCamera camera = mCamera;
4. DisplayItem[] displayItems = mDisplayItems;
5. int firstBufferedVisibleSlot = mBufferedVisibleRange.begin;
6. int lastBufferedVisibleSlot = mBufferedVisibleRange.end;
7. boolean isCameraZAnimating = mCamera.isZAnimating();

复制代码

**删除当前选中槽位前后2步之外的大缩略图**

1. for (int i = firstBufferedVisibleSlot; i <= lastBufferedVisibleSlot; ++i) {
2. if (selectedSlotIndex != Shared.INVALID && (i >= selectedSlotIndex - 2 && i <= selectedSlotIndex + 2)) {
3. continue;
4. }
5. DisplayItem displayItem = displayItems[(i - firstBufferedVisibleSlot) \* GridLayer.MAX\_ITEMS\_PER\_SLOT];
6. if (displayItem != null) {
7. displayItem.clearScreennailImage();
8. }
9. }

复制代码

**得到当前图片的DispalyItem元素**

1. int centerIndexInDrawnArray = (selectedSlotIndex - firstBufferedVisibleSlot) \* GridLayer.MAX\_ITEMS\_PER\_SLOT;
2. if (centerIndexInDrawnArray < 0 || centerIndexInDrawnArray >= displayItems.length) {
3. return;
4. }
5. DisplayItem centerDisplayItem = displayItems[centerIndexInDrawnArray];
6. if (centerDisplayItem == null || centerDisplayItem.mItemRef.mId == Shared.INVALID) {
7. return;
8. }

复制代码

**判断大缩略图是否加载完成**

1. boolean focusItemTextureLoaded = false;
2. Texture centerTexture = centerDisplayItem.getScreennailImage(view.getContext());
3. if (centerTexture != null && centerTexture.isLoaded()) {
4. focusItemTextureLoaded = true;
5. }

复制代码

**是否跳过当前图片前一张图片**

1. float camX = camera.mLookAtX \* camera.mScale;
2. float centerTranslateX = centerDisplayItem.mAnimatedPosition.x;
3. final boolean skipPrevious = centerTranslateX < camX;

复制代码

**开启opengl混合模式并设置混合函数**

1. gl.glEnable(GL11.GL\_BLEND);
2. gl.glBlendFunc(GL11.GL\_ONE, GL11.GL\_ONE);

复制代码

**循环遍历前中后三幅图片分别进行“核心绘制处理”**

1. for (int i = -1; i <= 1; ++i) {
2. if (slideshowMode && timeElapsedSinceView > 1.0f && i != 0)
3. continue;
4. 。。。
5. }

复制代码

**核心绘制处理——输入变量准备**

1. if (slideshowMode && timeElapsedSinceView > 1.0f && i != 0)
2. continue;
3. float viewAspect = camera.mAspectRatio;
4. int selectedSlotToUse = selectedSlotIndex + i;
5. if (selectedSlotToUse >= 0 && selectedSlotToUse <= lastBufferedVisibleSlot) {
6. int indexInDrawnArray = (selectedSlotToUse - firstBufferedVisibleSlot) \* GridLayer.MAX\_ITEMS\_PER\_SLOT;
7. if (indexInDrawnArray < 0 || indexInDrawnArray >= displayItems.length) {
8. return;
9. }
10. DisplayItem displayItem = displayItems[indexInDrawnArray];
11. MediaItem item = displayItem.mItemRef;
12. final Texture thumbnailTexture = displayItem.getThumbnailImage(view.getContext(), sThumbnailConfig);
13. Texture texture = displayItem.getScreennailImage(view.getContext());

复制代码

在幻灯模式下且超过1秒的切换时间无须显示前后两张图片；  
得到视角和当前displayItem、对应媒体对象、小缩略图材质、大缩略图材质。  
**加载高质量的材质资源**

1. if (isCameraZAnimating && (texture == null || !texture.isLoaded())) {
2. texture = thumbnailTexture;
3. mSelectedMixRatio.setValue(0f);
4. mSelectedMixRatio.animateValue(1f, 0.75f, view.getFrameTime());
5. }
6. Texture hiRes = (zoomValue != 1.0f && i == 0 && item.getMediaType() != MediaItem.MEDIA\_TYPE\_VIDEO) ? displayItem
7. .getHiResImage(view.getContext())
8. : null;
9. if (Gallery.PIXEL\_DENSITY > 1.0f) {
10. hiRes = texture;
11. }
12. if (i != 0) {
13. displayItem.clearHiResImage();
14. }
15. if (hiRes != null) {
16. if (!hiRes.isLoaded()) {
17. view.bind(hiRes);
18. view.prime(hiRes, true);
19. } else {
20. texture = hiRes;
21. }
22. }

复制代码

如果Camera正在拉远或拉近，且大缩略图材质为空或未加载完成，则选择小缩略图作为材质，将当前图片的“大缩略图混合比例”变量进行初始化（目标值为1秒，渐变时间为0.75秒，渐变开始时间为当前帧时间）。  
如果处于放大状态，则加载原图hiRes，加载成功后赋值给材质变量texture；并清除前后图片的原图。  
**加载材质**

1. final Texture fsTexture = texture;
2. if (texture == null || !texture.isLoaded()) {
3. if (Math.abs(centerTranslateX - camX) < 0.1f) {
4. if (focusItemTextureLoaded && i != 0) {
5. view.bind(texture);
6. }
7. if (i == 0) {
8. view.bind(texture);
9. view.prime(texture, true);
10. }
11. }
12. texture = thumbnailTexture;
13. if (i == 0) {
14. mSelectedMixRatio.setValue(0f);
15. mSelectedMixRatio.animateValue(1f, 0.75f, view.getFrameTime());
16. }
17. }

复制代码

保留当前的最佳材质，  
如果此材质加载未完成，则继续按优先级加载，并把小缩略图（基本上都已经加载成功了）设置为当前材质；  
最佳材质未加载完成之前，替换渐变不会开始。  
**无须绘制**

1. if (mCamera.isAnimating() || slideshowMode) {
2. if (!slideshowMode && skipPrevious && i == -1) {
3. continue;
4. }
5. if (!skipPrevious && i == 1) {
6. continue;
7. }
8. }
9. int theta = (int) displayItem.getImageTheta();

复制代码

如果相机缩放过程中，非幻灯片模式下且镜头中不需要展示前一张的情况下，无须处理前一张；  
如果相机缩放或幻灯片绘制过程中，需要展示前一张的情况下，无须处理后一张。  
**处理前后渐变绘制**

1. // If it is in slideshow mode, we draw the previous item in
2. // the next item's position.
3. if (slideshowMode && timeElapsedSinceView < 1.0f && timeElapsedSinceView != 0) {
4. if (i == -1) {
5. int nextSlotToUse = selectedSlotToUse + 1;
6. if (nextSlotToUse >= 0 && nextSlotToUse <= lastBufferedVisibleSlot) {
7. int nextIndexInDrawnArray = (nextSlotToUse - firstBufferedVisibleSlot)
8. \* GridLayer.MAX\_ITEMS\_PER\_SLOT;
9. if (nextIndexInDrawnArray >= 0 && nextIndexInDrawnArray < displayItems.length) {
10. float currentImageTheta = displayItem.mAnimatedImageTheta;
11. displayItem = displayItems[nextIndexInDrawnArray];
12. backupImageTheta = displayItem.mAnimatedImageTheta;
13. displayItem.mAnimatedImageTheta = currentImageTheta;
14. view.setAlpha(1.0f - timeElapsedSinceView);
15. }
16. }
17. } else if (i == 0) {
18. displayItem.mAnimatedImageTheta = backupImageTheta;
19. view.setAlpha(timeElapsedSinceView);
20. }
21. }

复制代码

如果处于幻灯片模式中 渐变过程中，处理上一幅相片时找到它的下一个DispalyItem，处理本张相片则直接使用当前DispalyItem，为的是同样找到当前DisplayItem，并绑定上一张相片和本张相片，两张相片透明度互补达到渐变的效果。  
**绘制小大缩略图渐变过程-小缩略图**

1. int vboIndex = i + 1;
2. float alpha = view.getAlpha();
3. float selectedMixRatio = mSelectedMixRatio.getValue(view.getFrameTime());
4. if (selectedMixRatio != 1f) {
5. texture = thumbnailTexture;
6. view.setAlpha(alpha \* (1.0f - selectedMixRatio));
7. }
8. GridQuad quad = GridDrawables.sFullscreenGrid[vboIndex];
9. float u = texture.getNormalizedWidth();
10. float v = texture.getNormalizedHeight();
11. float imageWidth = texture.getWidth();
12. float imageHeight = texture.getHeight();
13. boolean portrait = ((theta / 90) % 2 == 1);
14. if (portrait) {
15. viewAspect = 1.0f / viewAspect;
16. }
17. quad.resizeQuad(viewAspect, u, v, imageWidth, imageHeight);
18. quad.bindArrays(gl);
20. drawDisplayItem(view, gl, displayItem, texture, PASS\_FOCUS\_CONTENT, null, 0.0f);
21. quad.unbindArrays(gl);

复制代码

绘制小缩略图，请注意：selectedMixRatio表示大缩略图的绘制透明度，小缩略图的自然就是1.0f - selectedMixRatio。  
**绘制小大缩略图渐变过程-大缩略图**

1. if (selectedMixRatio != 0.0f && selectedMixRatio != 1.0f) {
2. texture = fsTexture;
3. if (texture != null) {
4. float drawAlpha = selectedMixRatio;
5. view.setAlpha(alpha \* drawAlpha);
6. u = texture.getNormalizedWidth();
7. v = texture.getNormalizedHeight();
8. imageWidth = texture.getWidth();
9. imageHeight = texture.getHeight();
10. quad.resizeQuad(viewAspect, u, v, imageWidth, imageHeight);
11. quad.bindArrays(gl);
12. drawDisplayItem(view, gl, displayItem, fsTexture, PASS\_FOCUS\_CONTENT, null, 1.0f);
13. quad.unbindArrays(gl);
14. }
15. }

复制代码

**更新当前图片长宽数据**

1. if (i == 0 || slideshowMode) {
2. mCurrentFocusItemWidth = quad.getWidth();
3. mCurrentFocusItemHeight = quad.getHeight();
4. if (portrait) {
5. // Swap these values.
6. float itemWidth = mCurrentFocusItemWidth;
7. mCurrentFocusItemWidth = mCurrentFocusItemHeight;
8. mCurrentFocusItemHeight = itemWidth;
9. }
10. }

复制代码

**绘制视频元素**

1. view.setAlpha(alpha);
2. if (item.getMediaType() == MediaItem.MEDIA\_TYPE\_VIDEO) {
3. // The play graphic overlay.
4. GridDrawables.sVideoGrid.bindArrays(gl);
5. drawDisplayItem(view, gl, displayItem, drawables.mTextureVideo, PASS\_VIDEO\_LABEL, null, 0);
6. GridDrawables.sVideoGrid.unbindArrays(gl);
7. }

复制代码

【此部分讲解已结束】，如您对其他部分感兴趣请回帖说明

[**gallery3d源码学习总结（二）——绘制流程drawThumbnails**](http://www.eoeandroid.com/thread-41731-1-1.html)

应大家要求，续写了第二部分，如有错误敬请赐教。  
此函数控制相册表格页、相片表格页、时间分类表格页的展示，非常重要。以下以相册表格页为例进行讲解，其他的就举一反三吧。  
**准备输入参数**

1. final GridDrawables drawables = mDrawables;
2. final DisplayList displayList = mDisplayList;
3. final DisplayItem[] displayItems = mDisplayItems;
4. final int firstBufferedVisibleSlot = mBufferedVisibleRange.begin;
5. final int lastBufferedVisibleSlot = mBufferedVisibleRange.end;
6. final int firstVisibleSlot = mVisibleRange.begin;
7. final int lastVisibleSlot = mVisibleRange.end;
8. final int selectedSlotIndex = mSelectedSlot;
9. final int currentFocusSlot = mCurrentFocusSlot;
10. final int currentScaleSlot = mCurrentScaleSlot;
11. final DisplayItem[] itemsDrawn = mItemsDrawn;
12. itemsDrawn[0] = null; // No items drawn yet.
13. int drawnCounter = 0;
14. final GridQuad grid = GridDrawables.sGrid;
15. grid.bindArrays(gl);
16. int numTexturesQueued = 0;
17. Context context = view.getContext();

复制代码

通过之前的computeVisibleItems()，已得到显示元素列表，显示元素缓冲数组，缓存可见槽位的范围，可见槽位的范围；得到当前选中的槽位索引selectedSlotIndex、当前焦点槽位索引currentFocusSlot和缩放槽位索引currentScaleSlot。  
已画元素的缓存数组，绑定表格类材质。selectedSlotIndex、currentFocusSlot、currentScaleSlot在不进行任何操作时都是-1.  
  
**遍历缓存可见槽位范围中的每一个槽位**

1. for (int itrSlotIndex = firstBufferedVisibleSlot; itrSlotIndex <= lastBufferedVisibleSlot; ++itrSlotIndex) {
2. int index = itrSlotIndex;
3. ...
4. }

复制代码

即遍历每个相册，index是相册序号。  
  
**为每个相册分别计算“需要绘制“的最开头那张照片的序号**

1. boolean priority = !(index < firstVisibleSlot || index > lastVisibleSlot);
2. int startSlotIndex = 0;
3. final int maxDisplayedItemsPerSlot = (index == mCurrentScaleSlot) ? GridLayer.MAX\_DISPLAYED\_ITEMS\_PER\_FOCUSED\_SLOT
4. : GridLayer.MAX\_DISPLAYED\_ITEMS\_PER\_SLOT;
5. if (index != mCurrentScaleSlot) {
6. for (int j = maxDisplayedItemsPerSlot - 1; j >= 0; --j) {
7. DisplayItem displayItem = displayItems[(index - firstBufferedVisibleSlot) \* GridLayer.MAX\_ITEMS\_PER\_SLOT + j];
8. if (displayItem == null) {
9. continue;
10. } else {
11. Texture texture = displayItem.getThumbnailImage(context, sThumbnailConfig);
12. if (texture != null && texture.isLoaded() == false) {
13. startSlotIndex = j;
14. break;
15. }
16. }
17. }
18. }

复制代码

priority表示当前槽位在[firstVisibleSlot,lastVisibleSlot]范围中，则为高优先级，可见槽位上的自然是最高优先级喽；  
maxDisplayedItemsPerSlot表示每个槽位中最多的显示项目，即每个相册封面的最大缩略图数，此处为4；  
startSlotIndex是当前帧 每个相册中“需要绘制“的最开头那张照片的序号（取值在0-3之间），注意“需要绘制“的最开头那张照片往往不是相册中的第一张照片，帧动画开始时它通常是第四张照片（第四张被依次压在321张照片下面嘛，因此绘制加载材质都要按照第四张向首张的次序），随着动画和材质逐步加载，它慢慢变为第三张、第二张、第一张。  
  
**从第四张向第一张加载材质**

1. // Prime the textures in the reverse order.
2. for (int j = 0; j < maxDisplayedItemsPerSlot; ++j) {
3. int stackIndex = (index == mCurrentScaleSlot) ? maxDisplayedItemsPerSlot - j - 1 : j;
4. DisplayItem displayItem = displayItems[(index - firstBufferedVisibleSlot) \* GridLayer.MAX\_ITEMS\_PER\_SLOT
5. + stackIndex];
6. if (displayItem == null) {
7. continue;
8. } else {
9. displayItem.mCurrentSlotIndex = index;
10. if (selectedSlotIndex != Shared.INVALID && (index <= selectedSlotIndex - 2 || index >= selectedSlotIndex + 2)) {
11. displayItem.clearScreennailImage();
12. }
13. Texture texture = displayItem.getThumbnailImage(context, sThumbnailConfig);
14. if (index == mCurrentScaleSlot && texture != null && !texture.isLoaded()) {
15. view.prime(texture, true);
16. view.bind(texture);
17. } else if (texture != null && !texture.isLoaded() /\*&& numTexturesQueued <= 6\*/) {
18. /\*boolean isCached = texture.isCached();\*/
19. view.prime(texture, priority);
20. view.bind(texture);
21. /\*if (priority && isCached && texture.mState != Texture.STATE\_ERROR)
22. ++numTexturesQueued;\*/
23. }
24. }
25. }

复制代码

因为是正序的处理4个元素，插入队头，因此是按照从第四张向第一张的次序加载材质。无用的代码已经注释掉了。  
  
**准备一些变量**

1. if (itrSlotIndex == selectedSlotIndex) {
2. continue;
3. }
4. view.prime(drawables.mTexturePlaceholder, true);
5. Texture placeholder = (state == GridLayer.STATE\_GRID\_VIEW) ? drawables.mTexturePlaceholder : null;
6. final boolean pushDown = (state == GridLayer.STATE\_GRID\_VIEW || state == GridLayer.STATE\_FULL\_SCREEN) ? false : true;

复制代码

加载占位材质，自己找找看是那个图标；如果是相片表格视图，则占位材质使用这个图标，否则占位材质为空；如果是相片表格视图，则长按选中后向上弹起，否则向下陷下去。  
  
**调整显示项目和相机的偏移量**

1. for (int j = 0; j < GridLayer.MAX\_ITEMS\_PER\_SLOT; ++j) {
2. DisplayItem displayItem = displayItems[(index - firstBufferedVisibleSlot) \* GridLayer.MAX\_ITEMS\_PER\_SLOT + j];
3. if (displayItem == null)
4. continue;
5. Texture texture = displayItem.getThumbnailImage(context, sThumbnailConfig);
6. if (texture == null || !texture.isLoaded()) {
7. if (currentScaleSlot != index) {
8. if (j == 0) {
9. final MediaSet parentSet = displayItem.mItemRef.mParentMediaSet;
10. if (parentSet != null && parentSet.getNumItems() <= 1) {
11. displayList.setAlive(displayItem, false);
12. }
13. } else {
14. displayList.setAlive(displayItem, false);
15. }
16. }
17. }
18. final float dx1 = mScaleGestureDetector.getTopFingerDeltaX();
19. final float dy1 = mScaleGestureDetector.getTopFingerDeltaY();
20. final float dx2 = mScaleGestureDetector.getBottomFingerDeltaX();
21. final float dy2 = mScaleGestureDetector.getBottomFingerDeltaY();
22. final float span = mScaleGestureDetector.getCurrentSpan();
23. if (state == GridLayer.STATE\_FULL\_SCREEN) {
24. displayList.setOffset(displayItem, false, true, span, dx1, dy1, dx2, dy2);
25. } else {
26. if (!mHoldPosition) {
27. if (state != GridLayer.STATE\_GRID\_VIEW) {
28. if (currentScaleSlot == index) {
29. displayList.setOffset(displayItem, true, false, span, dx1, dy1, dx2, dy2);
30. } else if (currentScaleSlot != Shared.INVALID) {
31. displayList.setOffset(displayItem, true, true, span, dx1, dy1, dx2, dy2);
32. } else {
33. displayList.setOffset(displayItem, false, false, span, dx1, dy1, dx2, dy2);
34. }
35. } else {
36. float minVal = -1.0f;
37. float maxVal = GridCamera.EYE\_Z \* 0.5f;
38. float zVal = minVal + mSpreadValue;
39. zVal = FloatUtils.clamp(zVal, minVal, maxVal);
40. if (Float.isInfinite(zVal) || Float.isNaN(zVal)) {
41. mCamera.moveZTo(0);
42. } else {
43. mCamera.moveZTo(-zVal);
44. }
45. }
46. }
47. }
48. }

复制代码

第一部分每太看懂，第二部分是得到多点触碰的输入信息，第三部分根据视图调整了照片偏移量和相机距离。  
  
**绘制缩略图显示项目**

1. for (int j = startSlotIndex; j < GridLayer.MAX\_ITEMS\_PER\_SLOT; ++j) {
2. DisplayItem displayItem = displayItems[(index - firstBufferedVisibleSlot) \* GridLayer.MAX\_ITEMS\_PER\_SLOT + j];
3. if (displayItem == null) {
4. break;
5. } else {
6. if (currentFocusSlot == index) {
7. displayList.setHasFocus(displayItem, true, pushDown);
8. mTargetFocusMixRatio = 1.0f;
9. } else {
10. displayList.setHasFocus(displayItem, false, pushDown);
11. }
12. if (j >= maxDisplayedItemsPerSlot)
13. continue;
14. Texture texture = displayItem.getThumbnailImage(view.getContext(), sThumbnailConfig);
15. if (texture != null) {
16. if (index == mCurrentScaleSlot)
17. displayItem.mAlive = true;
18. if ((!displayItem.isAnimating() || !texture.isLoaded())
19. && displayItem.getStackIndex() > GridLayer.MAX\_ITEMS\_PER\_SLOT) {
20. displayList.setAlive(displayItem, true);
21. continue;
22. }
23. if (index < firstVisibleSlot || index > lastVisibleSlot) {
24. if (view.bind(texture)) {
25. displayList.setAlive(displayItem, true);
26. }
27. continue;
28. }
29. drawDisplayItem(view, gl, displayItem, texture, PASS\_THUMBNAIL\_CONTENT, placeholder,
30. displayItem.mAnimatedPlaceholderFade);
31. } else {
32. // Move on to the next stack.
33. break;
34. }
35. if (drawnCounter >= GridLayer.MAX\_ITEMS\_DRAWABLE - 1 || drawnCounter < 0) {
36. break;
37. }
38. // Insert in order of z.
39. itemsDrawn[drawnCounter++] = displayItem;
40. itemsDrawn[drawnCounter] = null;
41. }
42. }

复制代码

如果是当前选中，则根据当前视图选择下陷还是弹起；绘制缩略图（其中的displayList.setAlive还没有看懂有什么用，如果有谁明白，愿闻其详）；记录已绘制的items。

[**gallery3d源码学习总结（三）——Cache缓存及数据处理流程**](http://www.eoeandroid.com/thread-41920-1-1.html)

第一，在应用程序中有三个线程存在：主线程（随activity的声明周期启动销毁）、feed初始化线程（进入程序时只运行一次，用于加载相册初始信息）、feed监听线程（一直在跑，监听相册和相片的变更）。  
  
第二，不考虑CacheService 启动的主要流程归纳如下：  
  
1.首次进入程序Gallery调用onCreate，此时发送初始化消息进入消息队列；然后Gallery调用onResume，向下进入GridLayer的onResume，如果此时Mediafeed对象没有进行初始化则什么也不干（第一次一般都是这样），否则调用Mediafeed的onResume；  
  
2.处理消息队列中的HANDLE\_INTENT消息，Gallery处理这个消息会初始化数据源，从而调用GridLayer的setDataSource方法，这个方法会触发底层MediaFeed的启动方法start，执行完后启动feed监听线程 继续执行MediaFeed的run方法。  
  
start方法会作两件事：调用自己底层的重新开始方法onResume，onResume中会为图像和视频这两个媒体源分别增加“内容变化监听器”，并请求马上刷新这两个媒体源（加入全局的刷新请求列表）；启动feed初始化线程mAlbumSourceThread。  
  
3.其中MediaFeed初始化线程的工作是：调用MediaFeed的loadMediaSets加载相册，它又调用了下层LocalDataSource中的refresh方法（查询数据库是否有相册变化，新增或修改feed中相应的MediaSet相册的名字）和loadMediaSets方法（调用下层CacheService.loadMediaSets方法，这个方法下面会重点展开）加载所有相册和相册中的所有相片信息。  
  
4.MediaFeed监听线程MediaFeed.run()的工作是：根据“内容变化监听器“返回的媒体变动消息（增删改），持续不断的更新MediaFeed中的相册和相片变量。  
  
具体机制是这样的，如果全局的刷新请求列表中有内容，则调用LocalDataSource.refresh进行相册信息的更新（其中LocalDataSource.refresh调用了CacheService的computeDirtySets），然后run遍历每个相册并调用dataSource.loadItemsForSet()方法为相册加载相片记录。  
  
第三，不考虑CacheService 新增和删除相片文件夹的主要流程归纳如下：  
  
1.对应【用相机照相后退回主程序，生成camera文件夹】部分，首先GridLayer判断mMediaFeed非空即已经初始化过，则直接调用MediaFeed.onResume方法，上面提到它会为图像和视频这两个媒体源分别增加“内容变化监听器”，并将其加入全局的刷新请求列表。  
  
2.可别忘了MediaFeed监听线程MediaFeed.run()方法，它一直在默默无闻的等待者全局的刷新请求列表有所变化。这时第二中的4这个步骤又被激活了，这之后更新相册和相片信息就是自然的了。  
  
3.对应【删除这个camera文件夹】，也是走的第2步这个过程。