**'Android 3D'에 해당되는 글 1건**

<http://aproject.tistory.com/tag/Android%203D>

1. 2009/08/07 [[안드로이드 3D 강좌 01] 3D 초기화](http://aproject.tistory.com/12) (1)

[**[안드로이드 3D 강좌 01] 3D 초기화**](http://aproject.tistory.com/12)

 | [Tutorials](http://aproject.tistory.com/category/Tutorials) 2009/08/07 10:30

Posted by fanta1102

우선 안드로이드 기본적으로 안드로이드 프로젝트를 생성할 수 있다는 전제하에 진행한다.

이 글은 anddev.org의 Android 2D/3D Graphics - OpenGL Tutorials을 바탕으로 한다.

(튜토리얼을 작성하고 소스를 오픈한 plusminus님에게 감사하며... ^----^)

그리고 강좌의 목적은 독학하기 위한것이므로 내용이 절대적이지 않다는 걸 명심하자!?

처음으로 해볼것은 바로 3D를 초기화해서 빈 화면을 그려보는 것이다.

일단 소스를 보도록 하자.

[view source](http://aproject.tistory.com/tag/Android%203D#viewSource)

[print](http://aproject.tistory.com/tag/Android%203D#printSource)[?](http://aproject.tistory.com/tag/Android%203D#about)

01.**package** com.android3d.tutorial01;

02.

03.**import** javax.microedition.khronos.egl.EGLConfig;

04.**import** javax.microedition.khronos.opengles.GL10;

05.

06.**import** android.app.Activity;

07.**import** android.opengl.GLSurfaceView;

08.**import** android.os.Bundle;

09.

10.**public** **class** Android3DTutorial01 **extends** Activity {

11.**private** GLSurfaceView mGLSurfaceView;

12.

13.@Override

14.**public** **void** onCreate(Bundle savedInstanceState) {

15.**super**.onCreate(savedInstanceState);

16.mGLSurfaceView = **new** GLSurfaceView(**this**);

17.mGLSurfaceView.setRenderer(**new** GLRenderer());

18.setContentView(mGLSurfaceView);

19.}

20.}

21.

22.**class** GLRenderer **implements** GLSurfaceView.Renderer {

23.

24.@Override

25.**public** **void** onDrawFrame(GL10 gl) {

26.gl.glClear(GL10.GL\_COLOR\_BUFFER\_BIT | GL10.GL\_DEPTH\_BUFFER\_BIT);

27.}

28.

29.@Override

30.**public** **void** onSurfaceChanged(GL10 gl, **int** width, **int** height) {

31.

32.}

33.

34.@Override

35.**public** **void** onSurfaceCreated(GL10 gl, EGLConfig config) {

36.gl.glClearColor(0, 0, 1, 1);

37.gl.glHint(GL10.GL\_PERSPECTIVE\_CORRECTION\_HINT, GL10.GL\_FASTEST);

38.}

39.}

안드로이드에서는 3D 그래픽을 그리기 위해서 OpenGL API를 사용한다.

엄밀히 말하자면 그냥 OpenGL API가 아닌 OpenGL ES API를 사용하는데

이것은 임베디드 기기에 좀 더 특화된 OpenGL API이라고 생각하면 된다.

J2ME에의 OpenGL ES API와 유사하지만 살짝쿵 다른 부분이 있을수도 있다(?)라고 가이드를 주고 있다.

자! 그럼 3D 그래픽을 위해서 준비해야 할 것들은 무엇인가?!

1. View 클래스를 정의하자!

2. OpenGLContext를 얻자!

3. Draw 메소드에서 GL 객체를 이용하여 정성을 다하여 그려보자.

먼저 View 클래스를 만드는 것에 대해 알아보자.

3D 그래픽이 필요한 목적이야 여러가지가 될 수 있지만 여기서는 게임과 같이

쉬지 않고 부지런히(?) 3D 화면을 그려줄 어플리케이션을 만든다는 전제를 깔아버리겠다.

게임에서 플레이어가 딴짓하며 캐릭터를 방치하든 조낸 몬스터를 쳐 때리고 있든

우리의 어플리케이션은 쉬지 않고 화면을 그려줄 것이다.

이러기 위해서 묵묵히 화면만 그려줄 쓰레드 하나가 필요하다.

이렇게 과묵히 자기 일을 수행할 서브 클래스들을 친절하게도 안드로이드는 이미 제공하고 있다.

SurfaceView나 GLSurfaceView가 바로 그런 클래스들이다.

이름을 봐도 딱 느낌이 오듯이 SurfaceView는 View의 자식이고 GLSurfaceView는 SurfaceView의 자식이다.

View와 SurfaceView는 android.view라는 패키지안에 같이 살고 있는 반면에

GLSurfaceView는 android.opengl라는 다른 패키지에 살림 차려 살고 있다.

대를 이어감에 따라 진화한다는 다윈옹의 진화론을 뒷받침이라도 하듯이

GLSurfaceView는 부모보다 상콤하게 깔끔한 모습으로 구현될 수 있다.

잡설이 길었다... 여차저차해서 결국 GLSurfaceView를 쓸것이다는 얘기다.

방법은 코드에서 보듯이 뷰를 설정할 때 GLSurfaceView 객체를 넣어주면 된다.

물론 이 상태에서도 어플리케이션은 문제 없이 돌아 간다.

하!지!만! 어디서 그려?? GLSurfaceView를 앙꼬빠진 호빵맨으로 만들지 않으려면

Renderer 객체를 만들어 설정해줘야 한다.

이 객체를 구현하기 위해서 GLSurfaceView.Renderer 인터페이스를 구현하도록 하면되는데

구색을 갖추려면 코드에서 보이는 세 가지 메소드가 반드시 정의되어야 한다.

실행화면은 다음과 같다.

|  |
| --- |
| http://cfile25.uf.tistory.com/image/143D6D124A7B836398CD37 |

[**[안드로이드 3D 강좌 02] 삼각형 그리기**](http://aproject.tistory.com/13)

 | [Tutorials](http://aproject.tistory.com/category/Tutorials) 2009/08/10 09:41

Posted by fanta1102

저번주에 두번째 강좌를 쓰려고 했으나 직교투영의 좌표가 제대로 먹히지 않아서 손 놓아버렸었다.  
에어컨이 고장나서 정신줄을 놓쳐 버릴것만 같았던 집에서의 주말을 보내고 나서  
냉장고에 들어온 듯한 느낌마저 드는 사무실에 앉으니 상콤하게 문제를 찾아버려  
다시 강좌를 시작한다. ^0^  
  
우선 간단하게 삼각형 하나 그려보자. 소스는 다음과 같다.

[view source](http://aproject.tistory.com/13#viewSource)

[print](http://aproject.tistory.com/13#printSource)[?](http://aproject.tistory.com/13#about)

01.**package** com.android3d.tutorial02;

02.

03.**import** java.nio.ByteBuffer;

04.**import** java.nio.ByteOrder;

05.**import** java.nio.FloatBuffer;

06.

07.**import** javax.microedition.khronos.egl.EGLConfig;

08.**import** javax.microedition.khronos.opengles.GL10;

09.

10.**import** android.app.Activity;

11.**import** android.opengl.GLSurfaceView;

12.**import** android.opengl.GLU;

13.**import** android.os.Bundle;

14.

15.**public** **class** Android3DTutorial02 **extends** Activity {

16.**private** GLSurfaceView mGLSurfaceView;

17.

18.@Override

19.**public** **void** onCreate(Bundle savedInstanceState) {

20.**super**.onCreate(savedInstanceState);

21.mGLSurfaceView = **new** GLSurfaceView(**this**);

22.mGLSurfaceView.setRenderer(**new** GLRenderer());

23.setContentView(mGLSurfaceView);

24.}

25.}

26.

27.**class** GLRenderer **implements** GLSurfaceView.Renderer {

28.**final** **static** **int** HVGA\_WIDTH = 320;      // HVGA portrait Width

29.**final** **static** **int** HVGA\_HEIGHT = 480;     // HVGA Portrait Height

30.

31.**float**[] triangle = **new** **float**[] {

32.-0.5f, 0, 0,

33.0, 0.5f, 0,

34.0.5f, 0, 0  };

35.

36.FloatBuffer triangleVB;

37.

38.**public** GLRenderer() {

39.triangleVB = createFloatBuffer(triangle);

40.}

41.

42.@Override

43.**public** **void** onDrawFrame(GL10 gl) {

44.gl.glClear(GL10.GL\_COLOR\_BUFFER\_BIT | GL10.GL\_DEPTH\_BUFFER\_BIT);

45.

46.gl.glColor4f(1, 0, 0, 0.5f);

47.gl.glVertexPointer(3, GL10.GL\_FLOAT, 0, triangleVB);

48.gl.glEnableClientState(GL10.GL\_VERTEX\_ARRAY);

49.gl.glDrawArrays(GL10.GL\_TRIANGLE\_STRIP, 0, 3);

50.}

51.

52.@Override

53.**public** **void** onSurfaceChanged(GL10 gl, **int** width, **int** height) {

54.

55.}

56.

57.@Override

58.**public** **void** onSurfaceCreated(GL10 gl, EGLConfig config) {

59.gl.glClearColor(0.0f, 0.0f, 1.0f, 1.0f);

60.

61.gl.glViewport(0, 0, HVGA\_WIDTH, HVGA\_HEIGHT);

62.gl.glMatrixMode(GL10.GL\_PROJECTION);

63.gl.glLoadIdentity();

64.GLU.gluOrtho2D(gl, -1.0f, 1.0f, -1.0f, 1.0f);

65.}

66.

67.**protected** FloatBuffer createFloatBuffer(**float**[] array) {

68.ByteBuffer byteBuffer = ByteBuffer.allocateDirect(array.length << 2);

69.byteBuffer.order(ByteOrder.nativeOrder());

70.FloatBuffer floatBuffer = byteBuffer.asFloatBuffer();

71.floatBuffer.put(array);

72.floatBuffer.position(0);

73.**return** floatBuffer;

74.}

75.}

OpenGL에서 정점을 정의하던 다음과 같은 방법은 더 이상 먹히지 않는다.  
  
glBegin (GL\_TRIANGLES);  
  glVertex3f (-1,0,0);  
  glVertex3f (0,1,0);  
  glVertex3f (1,0,0);  
glEnd();  
  
OpenGL ES는 명색이 임베디드를 위한 것이 아닌가!!  
정점을 정의할때 마다 디바이스 피곤하게 하지 않게 정점 정보를 배열로 만들고 한번에 처리를 한다.  
  
float[] triangle = new float[] {  
   -1, 0, 0,  
   0, 1, 0,  
   1, 0, 0 };  
  
하지만 안드로이드에서는 기본 실수형 배열을 전달할 수 없고 Buffer 클래스를 넘겨 줘야한다.  
이 작업을 해주는 createFloatBuffer() 메소드를 친절하게도 구현해 놓아서 감사하며 갖다 쓴다.  
이렇게 배열을 버퍼로 만들어 다음과 같이 삼각형을 화면에 그려낼 수 있다.  
  
gl.glVertexPointer(3, GL10.GL\_FLOAT, 0, triangleVB);  
gl.glEnableClientState(GL10.GL\_VERTEX\_ARRAY);  
gl.glDrawArrays(GL10.GL\_TRIANGLE\_STRIP, 0, 3);  
  
그리고 직교 투영이 제대로 되지 않았던 문제가 있었다.  
gl.Viewport() 메소드 없이 GLU.glOrtho2D() 메소드만 사용했을때 좌표계가 Y축으로 2만큼 이동되어 있었던 것이다.  
디폴트 값이 왜 그런지는 좀 더 고민해볼 문제이지만 당장은 중요하지 않다.  
이 문제는 뷰포트를 에뮬레이터의 해상도 맞춰주는 것으로 해결을 했다.  
  
실행화면은 다음과 같다.

|  |
| --- |
| http://cfile3.uf.tistory.com/image/11046D214A7F6C1A888DB1 |

### GLSurfaceView 소개

### <http://sites.google.com/site/endihom/home/programming-language/android/article/introducing-glsurfaceview>

|  |
| --- |
| android.opengl.GLSurfaceview 클래스는 아래와 같은 기능을 제공해 어플리케이션에서 OpenGL ES 렌더링을 쉽게 사용할수 있도록 해준다.   * [View](http://developer.android.com/reference/android/view/View.html) 시스템에 OpenGL ES를 연결할수 있는 연결코드를 제공한다. * OpenGL ES 작업이 [Activity](http://developer.android.com/reference/android/app/Activity.html) 라이프 싸이클과 함께 동작하도록 해주는 연결코드를 제공한다. * 적절한 프레임 버퍼 픽셀 형식을 쉽게 선택할수 있도록 해준다. * 부드러운 에니메이션이 가능하도록 분리된 렌더링 쓰레드를 생성하고 관리할수 있게 한다. * OpenGL ES API 호출 추적과 에러 체크를 위한 사용하기 쉬운 디버깅 툴을 제공한다.   GLSurfaceView는 부분적으로 혹은 전체적으로 렌더링에 사용하는 어플리케이션을 작성하기 위해 좋은 기반이 된다.[Google Map StreetView](http://www.youtube.com/watch?v=4PRfVKzuUJ4&fmt=18) 와 같은 2D, 3D 데이터 비쥬얼 어플리케이션과 같은 2D, 3D 액션 게임이 좋은 후보가 될 것이다.  단순한 GLSurfaceView 어플리케이션  가장 단순한 OpenGL ES 어플리케이션의 소스코드를 보자.  package com.example.android.apis.graphics;  import javax.microedition.khronos.egl.EGLConfig; import javax.microedition.khronos.opengles.GL10;  import android.app.Activity; import android.opengl.GLSurfaceView; import android.os.Bundle;  public class ClearActivity extends Activity {  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  mGLView = new GLSurfaceView(this);  mGLView.setRenderer(new ClearRenderer());  setContentView(mGLView);  }   @Override  protected void onPause() {  super.onPause();  mGLView.onPause();  }   @Override  protected void onResume() {  super.onResume();  mGLView.onResume();  }   private GLSurfaceView mGLView; }  class ClearRenderer implements GLSurfaceView.Renderer {  public void onSurfaceCreated(GL10 gl, EGLConfig config) {  // Do nothing special.  }   public void onSurfaceChanged(GL10 gl, int w, int h) {  gl.glViewport(0, 0, w, h);  }   public void onDrawFrame(GL10 gl) {  gl.glClear(GL10.GL\_COLOR\_BUFFER\_BIT | GL10.GL\_DEPTH\_BUFFER\_BIT);  } }  이 프로그램은 많은 일을 하지는 않는다. 모든 프레임에 검은색으로 화면을 지운다. 하지만 안드로이드 activity 라이프 싸이클을 정확하게 구현한 완벽한 OpenGL 어플리케이션이다.이 프로그램은 activity가 pause 상태일때 렌더링을 pause 하고 activity가 resume 될때 렌더링을 resume 한다. 이 프로그램을 기본적인 비상호적 데모 프로그램으로 사용할 수 있다.  ClearRenderer.onDrawFrame()메서드를 호출하기 위해 단순히 몇개의 OpenGL 호출을 추가했다. GLSurfaceview 를 상속할 필요가 없음에 주목하자.  GLSurfaceView.Render 인터페이스는 3가지 메소드를 가지는데   * onSurfaceCreate() 메서드는 렌더링의 시작 시점과 OpenGL ES 드로잉 context가 재생성될때마다 호출된다. (보통 드로잉 context는 activity가 pause나 resume 될때 소멸되어 다시 재생성된다.) OnSurfaceCreate()는 텍스쳐(역자 주 : 이미지를 OpenGL 객체에 입히는 작업)와 같은 생명 주기가 긴 OpenGL 리소스를 생성하기에 적절한 곳이다. * onSurfaceChanged() 메서드는 surface가 크기를 변경할 경우 호출된다. 이것은 OpenGL viewport를 설정하기에 적절한 곳이다. 장면주위를 움직이지 않는 고정된 카메라라면 여기서 카메라 설정을 할수 있다. * onDrawFrame() 메서드는 모든 프레임에서 호출되고 장면을 그리는 책임을 진다. 현재 장면을 그리기 위해 다른 OpenGL ES 호출 다음에 보통 프레임 버터를 삭제하기 위해 glClear를 호출할때 시작된다.  사용자 입력은 어떡해? 게임과 같은 상호 작용을 하는 어플리케이션을 원한다면 입력 이벤트를 받는 쉬운 방법을 제공받기 위해 GLSurfaceView를 상속해야한다. 어떻게 하는지 보여주는 조금 긴 예제를 보자.  package com.google.android.ClearTest;  import javax.microedition.khronos.egl.EGLConfig; import javax.microedition.khronos.opengles.GL10;  import android.app.Activity; import android.content.Context; import android.opengl.GLSurfaceView; import android.os.Bundle; import android.view.MotionEvent;  public class ClearActivity extends Activity {  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  mGLView = new ClearGLSurfaceView(this);  setContentView(mGLView);  }   @Override  protected void onPause() {  super.onPause();  mGLView.onPause();  }   @Override  protected void onResume() {  super.onResume();  mGLView.onResume();  }   private GLSurfaceView mGLView; }  class ClearGLSurfaceView extends GLSurfaceView {  public ClearGLSurfaceView(Context context) {  super(context);  mRenderer = new ClearRenderer();  setRenderer(mRenderer);  }   public boolean onTouchEvent(final MotionEvent event) {  queueEvent(new Runnable(){  public void run() {  mRenderer.setColor(event.getX() / getWidth(),  event.getY() / getHeight(), 1.0f);  }});  return true;  }   ClearRenderer mRenderer; }  class ClearRenderer implements GLSurfaceView.Renderer {  public void onSurfaceCreated(GL10 gl, EGLConfig config) {  // Do nothing special.  }   public void onSurfaceChanged(GL10 gl, int w, int h) {  gl.glViewport(0, 0, w, h);  }   public void onDrawFrame(GL10 gl) {  gl.glClearColor(mRed, mGreen, mBlue, 1.0f);  gl.glClear(GL10.GL\_COLOR\_BUFFER\_BIT | GL10.GL\_DEPTH\_BUFFER\_BIT);  }   public void setColor(float r, float g, float b) {  mRed = r;  mGreen = g;  mBlue = b;  }   private float mRed;  private float mGreen;  private float mBlue; }  이 어플리케이션은 모든 프레임을 위해 화면을 지운다. 사용자가 화면을 tap할때 사용자 touch 이벤트의 좌표값(x,y)에 기반해 색을 지운다. ClearGLSurfaceView.onTouchEvent()의 queueEvent()사용에 주목하자. queueEvent() 메서드는 UI 쓰레드와 렌더링 쓰레드간의 안전한 통신을 위해 사용된다. 원한다면 Renderer클래스 자체에서 동기화 메서드와 같은 자바 쓰레드간 통신기술을 사용할 수 있다. 하지만, 이벤트를 큐에 담는 것이 쓰레드간 통신을 다루는 것보다 더 안전한 방법이다. 다른 GLSurfaceView 샘플 단순히 화면을 지우는 것이 지루한가? 안드로이드 SDK에 포함된 API데모를 통해 더 재미있는 예제들을 볼수 있다. 모든 OpenGL ES 샘플들은 GLSurfaceView를 사용하기 위해 변경 되었다.   * GLSurfaceView - 삼각형 회전 * Kube - 큐브 퍼즐 데모 * 반투명 GLSurface 뷰 - 반투명 배경에서 3D 그래픽이 어떡해 디스플레이되는지를 보여준다. * 텍스쳐 삼각형 : 텍스쳐된 3D 삼각형을 어떡해 그리는지 보여준다. * 스프라이트 텍스트 - 텍스쳐에 텍스트를 그리고 3D 장면에 어떡해 구성되는지를 보여준다. * 터치 회전 : 사용자 입력에 반응하여 3D객체를 어떡해 회전시키는지를 보여준다.  surface 선택 GLSurfaceView는 렌더링하고자하는 surface의 유형을 선택할수 있도록 도와준다. 서로 다른 안드로이드 디바이스 공통 부분을 가지지 않는 서로 다른 Surface 유형을 지원한다. 이것은 각 디바이스별로 가장 적절한 Surface를 선택하는데 어려움을 준다.  디폴트로 GLSurfaceView는 16비트 깊이를 가지는 버퍼와 16비트 RGB 프레임 버퍼에 가능한 가까운 surface를 찾으려고 시도한다.  개발하고자하는 어플리케이션의 니즈에 따라 이 동작을 변경하고자 할수도 있을 것이다. 예를 들어 반투명 GLSurfaceView 샘플은 반투명 데이터를 렌더링 하기 위해 알파 채널을 필요로 할 것이다. GLSurfaceView는 어떤 surface 유형을 사용할지 선택할수 있도록 하기 위해 setEGLSurfaceChooser() 메서드를 재정의할수 있도록 한다.  setEGLConfigChooser(boolean needDepth)  *16비트 프레임 버퍼를 갖지 않을수 있는 R5G6B5에 가장 가까운 구성을 선택한다.*  setEGLConfigChooser(int redSize, int greenSize, int blueSize, int alphaSize, int depthSize, int stencilSize)  *적어도 생성자에서 명시된 것만큼의 채널당 비트수를 가지는 픽셀당 비트수가 가장 적은 구성을 선택한다.*  setEGLConfigChooser(EGLConfigChooser configChooser)  *구성 선택에 대한 총괄적인 제어를 가능하도록 해준다. 디바이스의 성능을 조사하여 구성을 선택할수 있는 EGLConfigChooser를 직접 구현하여 전달할수 있다.* 지속적인 렌더링 v.s 필요할때 렌더링 게임과 시뮬레이션과 같은 대부분의 3D 어플리케이션은 지속적으로 에니메이션된다. 그러나 몇몇 3D어플리케이션은 더 많은 반응을 하게되는데 사용자가 무언가를 수행할때까지 기다리고 그것에 대응하는 것과 같은 것이다. 이런 유형의 어플리케이션들위해 화면을 지속적으로 다시 그리는 디폴트 GLSurfaceView 동작은 시간 낭비이다. 반응하는 어플리케이션을 개발한다면 지속적인 에니메이션 하지는 GLSurfaceView.setRenderMode(RENDERMODE\_WHEN\_DIRTY) 를 호출 할 수 있고 다시 렌더링을 원할때는 GLSurfaceView.requestRender()를 호출 할 수 있다. 디버깅의 도움 GLSurfaceView는 OpenGL ES 어플리케이션 디버깅을 위해 유용한 내장 기능을 가지고 있다.GLSurface.setDebugFlags() 메서드는 OpenGL ES호출시 오류를 확인 하거나 로깅을 가능하게 하는데 사용될 수 있다. setRenderer()를 호출하기 전에  GLSurfaceView의 생성자에서 이메소드를 호출한다.  public ClearGLSurfaceView(Context context) {  super(context);  // 오류 확인과 로깅을 On   setDebugFlags(DEBUG\_CHECK\_GL\_ERROR | DEBUG\_LOG\_GL\_CALLS);  mRenderer = new ClearRenderer();  setRenderer(mRenderer); } |

[**GLSurfaceView 분석-I**](http://anddev.tistory.com/15)

<http://anddev.tistory.com/15>

[안드로이드](http://anddev.tistory.com/category/%EC%95%88%EB%93%9C%EB%A1%9C%EC%9D%B4%EB%93%9C) 2008/12/22 15:29

이번엔 안드로이드의 3D 샘플과 함께 제공되는 GLSurfaceView의 구현에 대해 분석해보겠습니다.

이 클래스는 OpenGL API를 사용하고 있기 때문에 OpenGL의 기본적인 내용은 알아야 합니다.

분석의 방법은...

제가 개인적으로 쓰는 방법입니다만...

보통 객체지향 언어는 캡슐화가 용이하다는 장점이 있는 반면

그에 대한 반대 급부로 코드의 해독이 난해해질 수 있다는 단점이 존재합니다.

따라서

분석 대상이 되는 코드에 친숙하지 않은 부분이 포함되어 있을 경우

캡슐화 되어 있는 내용을 풀어 Verbose하게 표현해보는 겁니다.

그리고 다시 역으로 거슬러 올라가면서

왜 이런 패턴을 적용하게 되었는지 원저자의 의도를 유추해내는 것이죠.

그래서 우선 GLSurfaceView 클래스가 하게 되는 일들을 시퀀셜하게 표현할 수 있도록

다음과 같이 풀어보았습니다.

|  |
| --- |
| private static class SampleView extends SurfaceView implements SurfaceHolder.Callback {  public static final String TAG = SampleView.class.getSimpleName();  private EGL10 mEgl;  private EGLDisplay mEglDisplay;  private EGLConfig mEglConfig;  private EGLContext mEglContext;  private EGLSurface mEglSurface;  public SampleView(Context context) {  super(context);  SurfaceHolder holder = getHolder();  holder.addCallback(this);  initOpenGL();  }  private void initOpenGL() {  mEgl = (EGL10) EGLContext.getEGL();  mEglDisplay = mEgl.eglGetDisplay(EGL10.EGL\_DEFAULT\_DISPLAY);  int[] version = new int[2];  mEgl.eglInitialize(mEglDisplay, version);  int[] configSpec = {  EGL10.EGL\_RED\_SIZE,      8,  EGL10.EGL\_GREEN\_SIZE,    8,  EGL10.EGL\_BLUE\_SIZE,     8,  EGL10.EGL\_ALPHA\_SIZE,    8,  EGL10.EGL\_DEPTH\_SIZE,   16,  EGL10.EGL\_NONE  };  int[] num\_config = new int[1];  EGLConfig[] configs = new EGLConfig[1];  mEgl.eglChooseConfig(mEglDisplay, configSpec, configs, 1,  num\_config);  mEglConfig = configs[0];  mEglContext = mEgl.eglCreateContext(mEglDisplay, configs[0],  EGL10.EGL\_NO\_CONTEXT, null);  if (mEglContext == null | mEgl.eglGetError() != EGL10.EGL\_SUCCESS)  {  Log.w(TAG, "Failed to get EGL context");  }  }  public void surfaceCreated(final SurfaceHolder holder) {  Log.v(TAG, "surfaceCreated");  mEglSurface = mEgl.eglCreateWindowSurface(mEglDisplay,  mEglConfig, holder, null);  mEgl.eglMakeCurrent(mEglDisplay, mEglSurface, mEglSurface,  mEglContext);  final GL10 gl = (GL10) mEglContext.getGL();  Log.v(TAG, "GL10: "+gl);  //뷰포트 설정  Rect size = getHolder().getSurfaceFrame();  gl.glViewport(0, 0, size.width(), size.height());  //좌표계 초기화  gl.glMatrixMode(GL10.GL\_PROJECTION);  gl.glLoadIdentity();  gl.glOrthof(-100, 100, -100, 100, 1, -1);  //배경화면 초기화  gl.glClearColor(0.5f, 0.5f, 0.5f, 1f);  gl.glClearDepthf(1.0f);  gl.glClear(GL10.GL\_COLOR\_BUFFER\_BIT | GL10.GL\_DEPTH\_BUFFER\_BIT);  Log.v(TAG, "clear color");  //모델링  gl.glMatrixMode(GL10.GL\_MODELVIEW);  gl.glLoadIdentity();  draw(gl);  mEgl.eglSwapBuffers(mEglDisplay, mEglSurface);  Log.v(TAG, "swap buffer");  };  private void draw(GL10 gl) {  //여기에서 실제 그리기 작업이 시작됩니다.  }  public void surfaceChanged(SurfaceHolder holder, int format, int width,  int height) {  Log.v(TAG, "surfaceChanged");  GL10 gl = ((GL10) mEglContext.getGL());  gl.glViewport(0, 0, width, height);  }  public void surfaceDestroyed(SurfaceHolder holder) {  Log.v(TAG, "surfaceDestroyed");  mEgl.eglMakeCurrent(mEglDisplay, EGL10.EGL\_NO\_SURFACE, EGL10.EGL\_NO\_SURFACE,mEglContext);  mEgl.eglDestroyContext(mEglDisplay, mEglContext);  mEgl.eglTerminate(mEglDisplay);  mEglSurface = null;  mEglContext = null;  mEglDisplay = null;  }  } |

다음 글에서 실제 GLSurfaceView의 디자인에 대해 살펴보도록 하겠습니다. 위의 내용들이 어떻게 클래스로 분리되어 표현되는지 눈여겨 보시면 도움이 될겁니다.

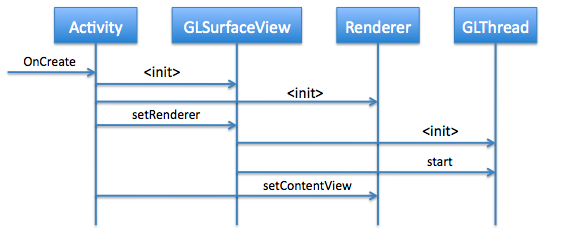
[**GLSurfaceView 분석-II**](http://anddev.tistory.com/18)

[안드로이드](http://anddev.tistory.com/category/%EC%95%88%EB%93%9C%EB%A1%9C%EC%9D%B4%EB%93%9C) 2008/12/24 17:34

GLSurfaceView는 SurfaceView의 하위 클래스이며 내부적으로 OpenGL 요청을 처리하기 위한 기능을 담고 있습니다. GLSurfaceView에는 다음과 같이 1개의 인터페이스와 2개의 클래스를 내부에 가지고 있습니다.

* 인터페이스
  + Renderer: 각 Frame에 그림을 그립니다.
* 클래스
  + GLThread: Renderer를 주기적으로 호출합니다.
  + EGLHelper: EGL 설정을 담당합니다.

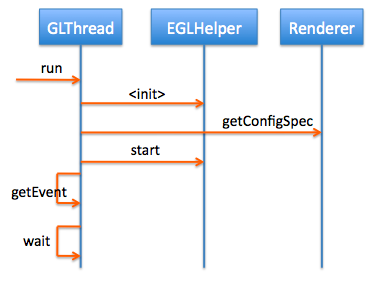
하나의 Activity에 하나의 GLSurfaceView가 존재하는 상황을 가정하겠습니다. 해당 Activity가 생성될 때 다음그림과 같은 과정이 수행됩니다. 푸른 색의 화살표는 Main 쓰레드를 뜻합니다.



1. Activity의 OnCreate 콜백 메소드가 호출됩니다.
2. OnCreate메소드는 GLSurfaceView와 Renderer 객체를 생성하고 GLSurfaceView 객체에 할당합니다.
3. GLSurfaceView는 GLThread를 생성하고 Renderer객체를 할당합니다.
4. GLThread를 start 합니다.
5. 생성된 GLSurfaceView를 Activity의 컨텐트 뷰로 설정합니다.

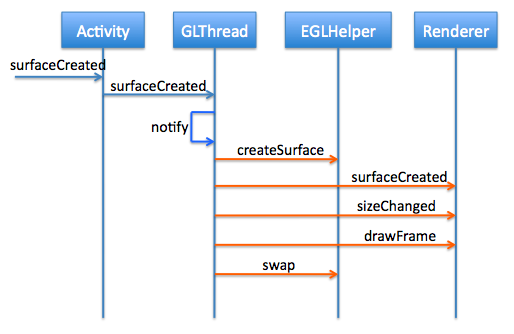
이제 GLThread는 별도의 쓰레드로 동작하게 됩니다. GLThread는 루프를 돌며 주기적으로 Renderer에게 화면 그리기를 요청합니다.  구체적인 동작은 다음과 같습니다.

GLThread는 Thread 클래스의 하위 클래스입니다. Thread클래스의 start메소드를 호출하면 새로운 쓰레드가 생성된 후 run메소드에 정의된 동작들이 수행됩니다. 아래의 그림은 쓰레드가 생성된 이후의 과정을 보여줍니다. 주황색 화살표는 GLThread를 뜻합니다.



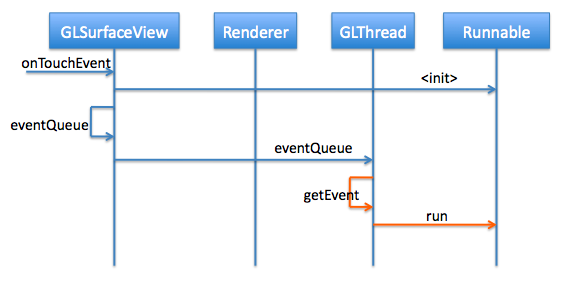
1. GLThread는 EGLHelper를 생성합니다.
2. GLThread는 EglContext생성을 위한 Configuration을 Renderer에게 질의합니다.
3. Renderer의 Configuration에 상응하는 EglContext와 EglDisplay가 생성됩니다.
4. 이벤트 큐를 확인한 후 아무것도 없으면 wait 상태에 들어갑니다. (반드시 wait가 호출되는 것은 아니며 내부 상태 변수에 따라 동작이 변할 수 있으나 여기에서는 상황을 단순화하여 표현하기 위해 이렇게 표현함)

다음 그림은 Surface가 생성된 후 SurfaceHolder.Callback 인터페이스의 surfaceCreated가 호출된 경우를 보여줍니다.



1. GLThread의 surfaceCreated 메소드를 호출. 메소드 내부에서 GLThread를 notify합니다.
2. notify된 GLThread가 EGLHelper에게 createSurface를 호출합니다.
3. Surface가 생성되었음을 Renderer에게 알립니다.
4. Renderer에게 drawFrame을 요청합니다.
5. 다 그린후 버퍼를 스왑하여 화면에 표시합니다(더블 버퍼링).

GLThread는 내부적으로 EventQueue를 가지고 있습니다. 동작 방식을 우선 살펴본 후 EventQueue가 왜 존재하는지를 고려해보겠습니다. 아래의 그림은 사용자가 스크린을 터치하여 해당 이벤트가 GLSurfaceView에 잡힌 이후의 상황을 보여줍니다.



1. Touch 이벤트에 대응되는 동작을 기술한 Runnable 객체를 이벤트 큐에 삽입합니다.
2. GL 쓰레드는 이벤트 큐를 검사하고 이벤트가 존재할 경우 해당 이벤트를 실행합니다.
3. 이후 여러 과정을 거친 후, Renderer의 drawFrame을 호출합니다.

GLThread의 EventQueue는 왜 존재할까요? 이것 역시 일반적인 UI모델을 흉내낸 것입니다. Eclipse나 Swing의 UI 툴킷은 Single Threaded 모델을 채택하고 있습니다. UI의 무분별한 갱신을 막고 일관성있는 화면 갱신 메카니즘을 제공하기 위한 것입니다. 그리고 사용자의 비동기적인 요구에 대응하기 위해 이벤트 큐를 유지하며 사용자의 요구는 이벤트 큐에 일단 저장되었다가 적절한 시점에 일괄적으로 처리합니다. 마찬가지로 GLSurfaceView도 화면 갱신을 위해서 단일 GLThread를 갖고 있으며 화면에 대한 추가적인 요구는 eventQueue를 이용하게 되어 있습니다. 그리고 또 하나의 특징은 푸른색의 화살표는 메인 쓰레드로서 여기에서 롱타임 연산이 발생할 경우 전체 UI의 멈춤 현상이 발생하게 됩니다. 따라서 롱타임 연산은 Runnable인터페이스로 감싸서 이벤트 큐에 등록하고 GLThread는 적절한 시점에 해당 연산을 수행하도록 함으로 전체 UI의 응답성을 훼손하지 않게 됩니다.

단점은 현재의 구현에서 Frame rate을 적절히 제어하는 코드가 없다는 점입니다. 이런 상황에서는 빠른 디바이스에서는 빠른 애니메이션을, 느린 디바이스에서는 느린 애니메이션을 보여주게 됩니다. 따라서 디바이스에 상관없이 일정한 Frame rate을 보여주게 하는 기능을 추가해야 할 것 같습니다. 이 부분에 대한 내용은 나중에 다시 다루기로 하지요.

**OpenGL 로 원하는 도형 만들기**

<http://hyena0.tistory.com/365>

  android로 OpenGL ES 를 이용하는 예제는

  Dev site 에서 찾을 수 있습니다.

  예제는 Cube.java 와 CubeRenderer.java 입니다.

  그런데 여기서 Cube 만 다루다 보니

  다른 도형을 어떻게 해야 할지 좀 막막하더군요.

  Web 상의 OpenGL 문서를 뒤져보고 해도 기초만 나오기 때문에

  좀처럼 이해하기 어려웠습니다.

  원하는 도형을 한번에 만들어서 Renderer 에서 그리면 좋을텐데,

  꼭지점을 이용하여 그리는 방식을 이해하기 어렵더군요.

  그간 고민한게 효과가 있었던지, 드디어 알게 되었습니다.

  일단 Cube.java 파일을 봅시다.

class Cube

{

    public Cube()

    {

        int one = 0x10000;

        int vertices[] = {

               //x,    y,    z,

                -one, -one, -one, // index #0

                one, -one, -one,  // index #1

                one,  one, -one,  // index #2

                -one,  one, -one, // index #3

                -one, -one,  one, // index #4

                one, -one,  one,  // index #5

                one,  one,  one,  // index #6

                -one,  one,  one, // index #7

        };

        int colors[] = {

              //R, G, B, A

                0,    0,    0,  one,

                one,    0,    0,  one,

                one,  one,    0,  one,

                0,  one,    0,  one,

                0,    0,  one,  one,

                one,    0,  one,  one,

                one,  one,  one,  one,

                0,  one,  one,  one,

        };

        byte indices[] = {

                // Triangle 로 쪼개는 인덱스 번호들

                0, 4, 5,    0, 5, 1,

                1, 5, 6,    1, 6, 2,

                2, 6, 7,    2, 7, 3,

                3, 7, 4,    3, 4, 0,

                4, 7, 6,    4, 6, 5,

                3, 0, 1,    3, 1, 2,

        };

        // Buffers to be passed to gl\*Pointer() functions

        // must be direct, i.e., they must be placed on the

        // native heap where the garbage collector cannot

        // move them.

        //

        // Buffers with multi-byte datatypes (e.g., short, int, float)

        // must have their byte order set to native order

        ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length\*4);

        vbb.order(ByteOrder.nativeOrder());

        mVertexBuffer = vbb.asIntBuffer();

        mVertexBuffer.put(vertices);

        mVertexBuffer.position(0);

        ByteBuffer cbb = ByteBuffer.allocateDirect(colors.length\*4);

        cbb.order(ByteOrder.nativeOrder());

        mColorBuffer = cbb.asIntBuffer();

        mColorBuffer.put(colors);

        mColorBuffer.position(0);

        mIndexBuffer = ByteBuffer.allocateDirect(indices.length);

        mIndexBuffer.put(indices);

        mIndexBuffer.position(0);

    }

    public void draw(GL10 gl)

    {

        gl.glFrontFace(gl.GL\_CW);

        gl.glVertexPointer(3, gl.GL\_FIXED, 0, mVertexBuffer);

        gl.glColorPointer(4, gl.GL\_FIXED, 0, mColorBuffer);

        gl.glDrawElements(gl.GL\_TRIANGLES, 36, gl.GL\_UNSIGNED\_BYTE, mIndexBuffer);

    }

    private IntBuffer   mVertexBuffer;

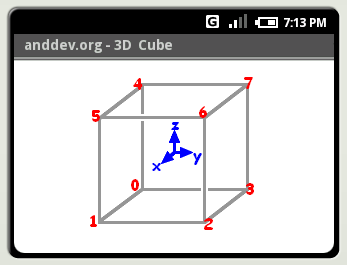
    private IntBuffer   mColorBuffer;

    private ByteBuffer  mIndexBuffer;

}

그리고 anddev.org 사이트에서 찾아보면 나오는 설명에서 아래의 그림처럼

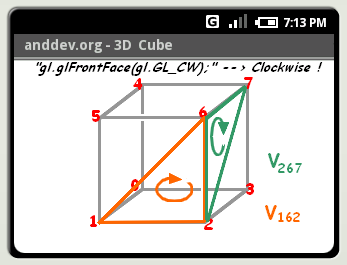
꼭지점들(vertices)을 정하고, 색을 정하고, 그림을 그리기 위한 조각들을 결정합니다.



위의 그림처럼 꼭지점이 인덱스가 "0" 이면 (-1,-1,-1) 이고,  "1"이면 (1,-1,-1)  의 순으로 지정됩니다.

vertices[] 내의 좌표 순서가 바로 인덱스 순서라고 보면 되지요.

그리고 glDrawElements 함수로 그리기 위해 GL\_TRIANGLES를 이용했는데 아래 그림처럼 삼각형으로 잘라지지요.

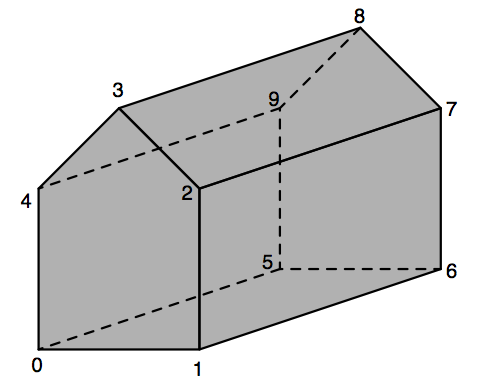


삼각형의 번호는 인덱스의 번호로 3개씩 짝을 짓고 있고 glFrontFace 함수에서 시계방향을 선택했으므로

인덱스 번호의 순서를 결정할 수 있습니다.

그러면 다른 도형의 형태를 어떻게 할 것인가? 감이 오시는지...

안오신다면 아래의 그림을 다시 한번 보면 이해가 될 것 같네요.



만약 집모양을 만든다고 하면 좌표값의 크기를 정하고, 좌표값에 맞는 인덱스를 정한 후

도형의 면을 그릴 수 있도록 삼각형, Loop 등의 방법을 결정해 주면 됩니다.

[**Blender로 도형만들기**](http://hyena0.tistory.com/417)

[**[iPhone] Blender 로 3D 도형 만들기 .. 1**](http://hyena0.tistory.com/417)

**Blender 로 3D 도형 만들기 .. 1**

  Blender 로 3D 도형을 만들고

  아이폰에서 OpenGL 로 보이는 것을 쓰겠다고

  해놓고선 한참지났네요.

  앱의 마케팅을 하느라 해봤지만, 역시나

  8만개나 되는 앱들에 묻혀서 다운수가 늘지가 않는군요..^^;

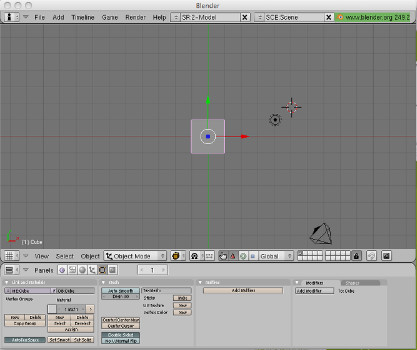
  지금의 Blender 로 만드는 도형의 방법은 안드로이드에서도

  동일하게 적용하시면 됩니다. 단, 출력된 값을 정수형으로 변경해야

  하는 것만 유념하면 될 것 같습니다. 아직 시험은 안해봤지만요...

  Blender 는 무료 공개 소프트웨어로 검색하시면 쉽게 다운로드 받을 수 있습니다.

  실행하면 아래와 같습니다.



  기본으로 정육면체가 만들어져 있습니다.

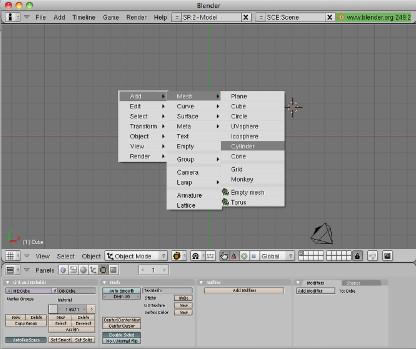
  Blender 웹페이지에 사용법이 있으니 여기서는 자세한 사용법보다는 아이폰에서 사용하기 위한

  용도위주로 설명해보겠습니다.

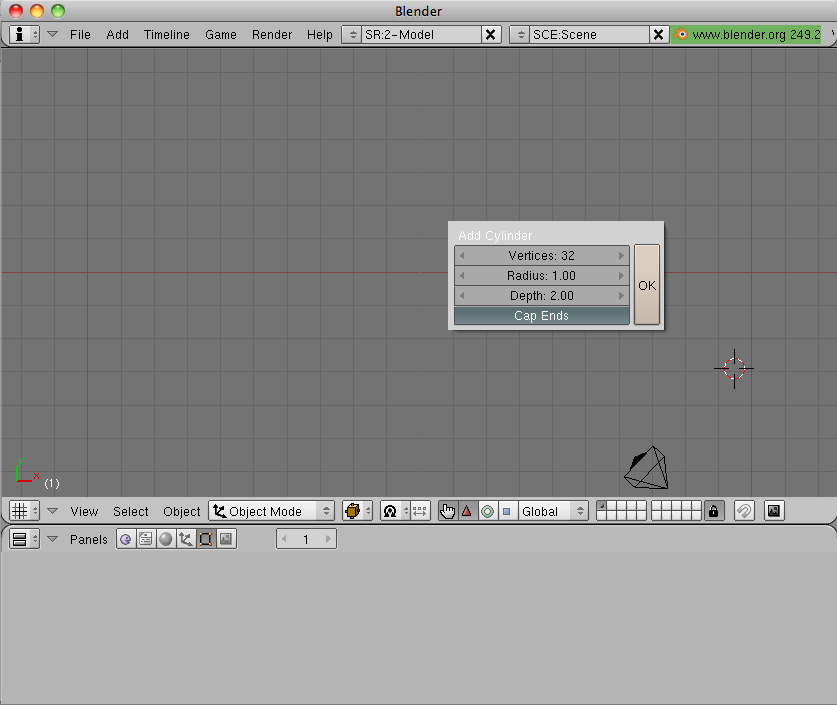
  Blender 의 사용법은 약간 기존의 마우스와 키보드 사용법과 다르기 때문에 생소할 수 있습니다.

  스페이스바를 누르면 메뉴가 나오게 되고 "Add"를 선택하면 기본 도형을 만들 수 있습니다.

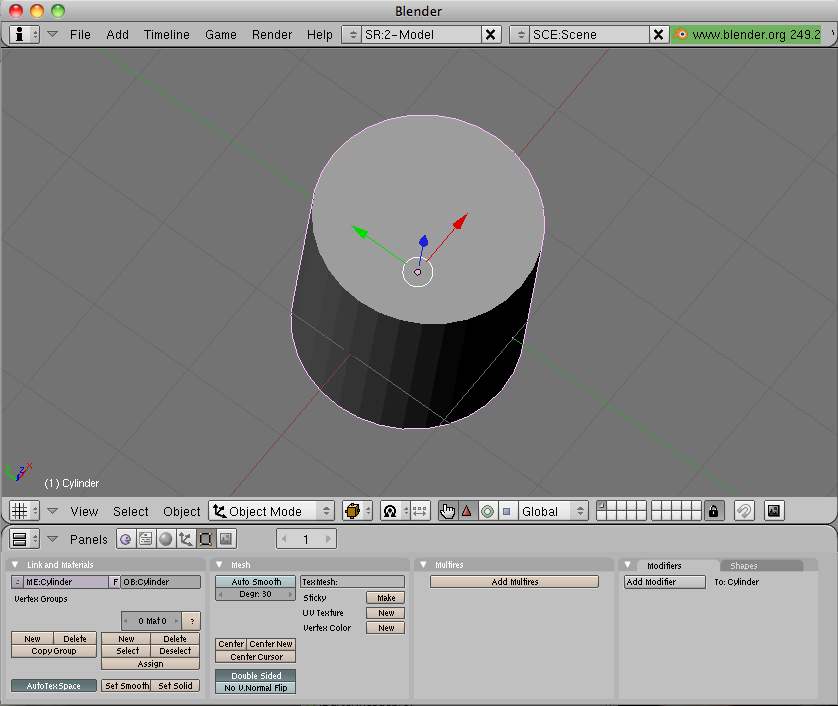
  원통을 만들어 보기로 하죠. Add>Cylinder 를 선택합니다.



  cylinder 를 선택하고 나면 아래와 같이 추가메뉴가 나오는데 반경과 높이 등을 설정하고 나면 도형이 생성됩니다.



도형은 수직에서 쳐도보도록 되어 있는데, 마우스의 볼을 누르고 움직이면 좌표가 회전되는 것을 볼 수 있습니다.



여기서 Edit 모드를 선택하면 꼭지점과 모서리가 선택되어 보여집니다.

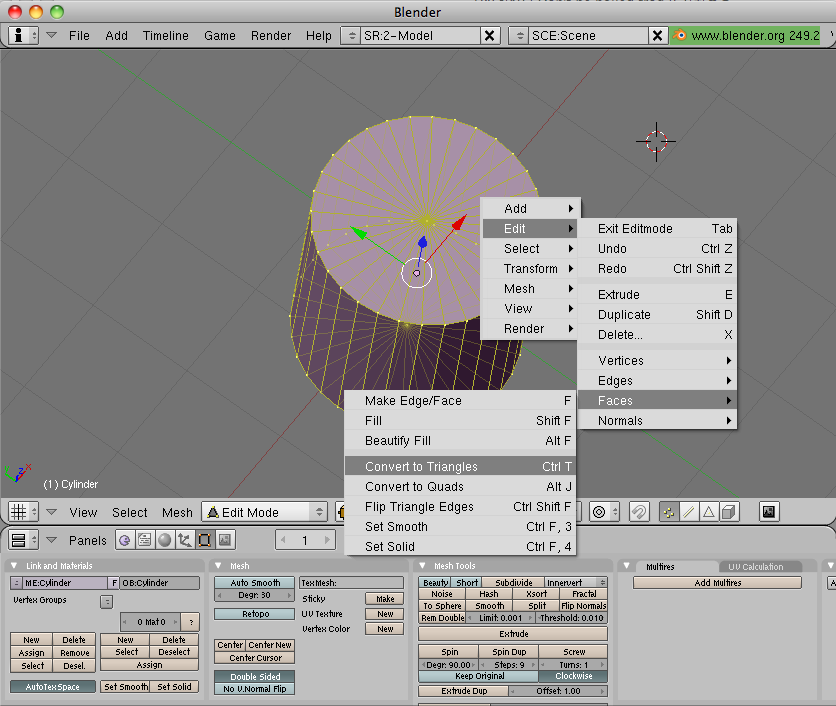
아래에서 보듯이 원통에서 원부분은 삼각형으로 나누어져 있고, 통부분은 사각형으로 만들어져 있습니다.

이렇게 되어 있으면 도형의 값을 추출했을때 OpenGL 로 변경하면 이상한 모양으로 변경됩니다.

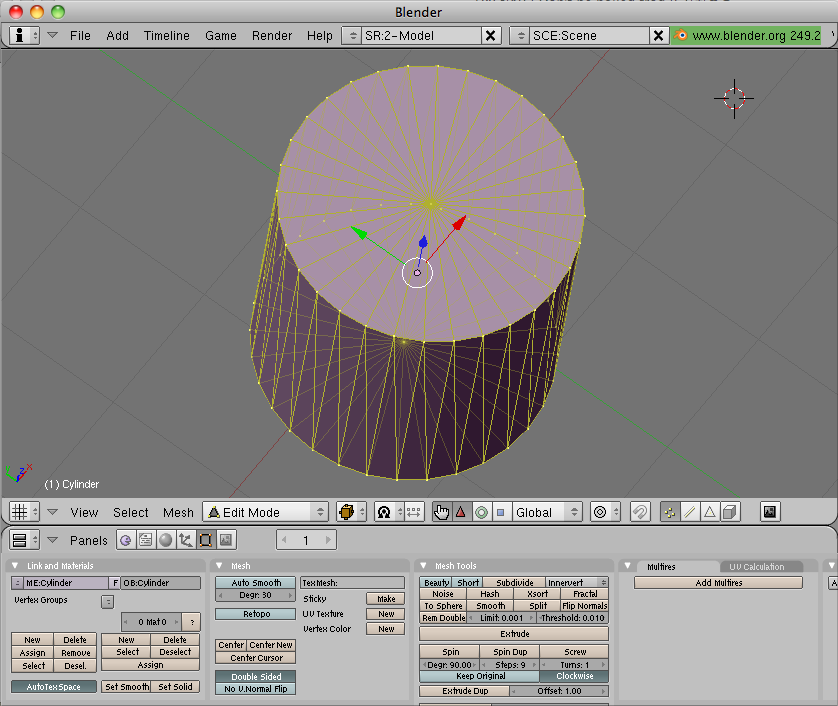
OpenGL에서 도형을 만들때는 삼각형을 이용하기 때문입니다.

그래서 삼각형으로 도형을 다시 잘라내야 합니다.

Edit>Faces>Convert to Triangles 를 선택하면 도형이 삼각형으로 다시 잘라집니다.



아래를 보시면 도형의 모양이 삼각형으로 되어 있는 것을 볼 수 있습니다.



  다음 편에서는 이렇게 만든 도형을 데이터로 뽑는 과정을 정리해 보겠습니다.

**[다음 : Blender 로 3D 도형만들기.. 2](http://hyena0.tistory.com/418" \o "[http://hyena0.tistory.com/418]로 이동합니다." \t "_blank)**

<http://blog.naver.com/sei2001?Redirect=Log&logNo=122314871>

|  |
| --- |
| **1. 안드로이드 3D 게임 튜토리얼(구글 번역기 버전)** http://blogimgs.naver.com/imgs/nblog/spc.gif [안드로이드 3D](http://blog.naver.com/PostList.nhn?blogId=sei2001&categoryNo=58) / [스마트폰 삼매경](http://blog.naver.com/PostList.nhn?blogId=sei2001&categoryNo=57&parentCategoryNo=57) http://blogimgs.naver.com/imgs/nblog/spc.gif  2011/02/16 02:12  복사<http://blog.naver.com/sei2001/122314871>  [첨부파일 (**1**)](http://blog.naver.com/PostView.nhn?blogId=sei2001&logNo=122314871&beginTime=0&jumpingVid=&from=search&redirect=Log&widgetTypeCall=true&topReferer=http%3A%2F%2Fsearch.naver.com%2Fsearch.naver%3Fwhere%3Dnexearch%26query%3D%25BE%25C8%25B5%25E5%25B7%25CE%25C0%25CC%25B5%25E5%2B3d%2B%25B0%25B3%25B9%25DF%26x%3D0%26y%3D0%26sm%3Dtop_hty%26fbm%3D2) |

안드로이드 공부겸 번역을 시작했으나 결국 귀차니즘으로 구글번역기로 돌림

원문:<http://www.droidnova.com/android-3d-game-tutorial-part-i,312.html>  
작성자:마틴(Matin)

첫번째 강좌는 안드로이드 2.0.1 환경에서 OpenGL를 이용한 3D 프로그래밍을 소개할 것 입니다.

이 강좌는 Vortex라는 게임을 바탕으로 3D프로그래밍에 중점을 두고 진행되며

기본적인 자바 개발 능력과 안드로이드 개발 지식이 있어야 이해할 수 있습니다.

먼저 OpenGL의 기본 용어입니다.  
Vertex(점) - 둘 이상의 선이 만나거나 교차하는 점 (X,Y,Z의 좌표를 가짐)  
Triangle(삼각형) - 말 그대로 삼각형(세개 점으로 만든 도형).  
Polygon(다각형) - 말 그대로 다각형(세개 이상의 점으로 만든 도형. 삼각형도 다각형에 포함).  
Primitives(원시도) - 삼각형 또는 다각형의 집합으로 만들어진 입면체입니다.

이제 소스를 분석하면서 3D 프로그래밍을 연습해보겠습니다.

먼저 Vortex 프로젝트를 생성하고 **Vortex**클래스를 아래와 같이 코딩합니다.

1 :  package com.droidnova.android.games.vortex;                              
2 :                                                                            
3 :  import android.app.Activity;                                              
4 :  import android.os.Bundle;                                                 
5 :                                                                            
6 :  public class **Vortex**extends Activity {                                    
7 :      private static final String LOG\_TAG = Vortex.class.getSimpleName();   
8 :      private **VortexView** \_vortexView;                                       
9 :                                                                            
10:      @Override                                                             
11:      protected void onCreate(Bundle savedInstanceState) {                  
12:          super.onCreate(savedInstanceState);                               
13:          \_vortexView = new **VortexView** (this);                               
14:          setContentView(\_vortexView);                                      
15:      }                                                                     
16:  }

**Vortex**클래스코드 8번째줄의 **VortexView** 클래스를 아래와 같이 코딩합니다.

1 :  package com.droidnova.android.games.vortex;                                 
2 :                                                                               
3 :  import android.content.Context;                                              
4 :  import android.opengl.GLSurfaceView;                                         
5 :                                                                               
6 :  public class **VortexView** extends **GLSurfaceView**{                              
7 :      private static final String LOG\_TAG = VortexView.class.getSimpleName();  
8 :      private VortexRenderer \_renderer;                                        
9 :                                                                               
10:      public VortexView(Context context) {                                     
11:          super(context);                                                      
12:          \_renderer = new VortexRenderer();                                    
13:          setRenderer(\_renderer);                                              
14:      }                                                                        
15:  }

VortexView 클래스는 그리는 기능을 담당하기 때문에 **GLSurfaceView**를 상속받습니다.

다음으로 3D 프레임 처리를 담당하는 **VortexRenderer**클래스를 아래와 같이 코딩 합니다.

 1: package com.droidnova.android.games.vortex;                                      
 2:                                                                                  
 3: import javax.microedition.khronos.egl.EGLConfig;                                 
 4: import javax.microedition.khronos.opengles.GL10;                                 
 5:                                                                                  
 6: import android.opengl.GLSurfaceView;                                             
 7:                                                                                  
 8: public class **VortexRenderer**implements **GLSurfaceView**.Renderer {                  
 9:     private static final String LOG\_TAG = VortexRenderer.class.getSimpleName();  
10:                                                                                  
11:     private float \_red = 0.9f;                                                   
12:     private float \_green = 0.2f;                                                 
13:     private float \_blue = 0.2f;                                                  
14:                                                                                  
15:     @Override                                                                    
16:     public void **onSurfaceCreated**(GL10 gl, EGLConfig config) {                    
17:         // Do nothing special.                                                   
18:     }                                                                            
19:                                                                                  
20:     @Override                                                                    
21:     public void **onSurfaceChanged**(GL10 gl, int w, int h) {                        
22:         gl.glViewport(0, 0, w, h);                                               
23:     }                                                                            
24:                                                                                  
25:     @Override                                                                    
26:     public void **onDrawFrame**(GL10 gl) {                                           
27:         // define the color we want to be displayed as the "clipping wall"       
28:         gl.**glClearColor**(\_red, \_green, \_blue, 1.0f);                              
29:         // clear the color buffer to show the ClearColor we called above...      
30:         gl.glClear(GL10.GL\_COLOR\_BUFFER\_BIT);                                    
31:     }                                                                            
32: }

**VortexRenderer**클래스는 **GLSurfaceView**.Renderer를 implements 하여

**onSurfaceCreated**()와  **onSurfaceChanged**(), **onDrawFrame**()를 재정의 합니다.

 **onSurfaceCreated**()는 이름에서도 알 수 있듯이 Surface가 처음 그려질 때 호출되며

**onSurfaceChanged**()는 Surface 설정값이 변경되면 호출되며

**onDrawFrame**()는 3D 프레임 처리시 호출됩니다.

11줄~13줄은 RGB칼라 시스템 값을 설정한 것입니다.

28줄 코드는 **glClearColor**()에 "**Clipping Wall**"의 색깔값을 정의한 것입니다.

**Clipping Wall**이란 **배경**을 말하는 것으로써 물체 **뒤**에 위치한다.  어떤 물체를 사라지게 할 때 **Clipping Wall**뒤로 보내면 된다.

또한**Clipping Wall**을 조절함으로써 안개같은 효과도 적용할 수 있다.

다음으로 **MotionEvent(동작이벤트)**에 따라 색을 변경시키는 기능을 코딩하겠습니다.  
다음과 같이 **VortexRenderer**클래스 안에 아래 메소드를 추가합니다.

1:  public void **setColor**(float r, float g, float b) {  
2:     \_red = r;                                       
3:     \_green = g;                                     
4:     \_blue = b;                                      
5: }

그리고 **VortexView** 클래스에서는 터치 이벤트가 발생할 경우 변경된 색깔이 적용되도록 **MotionEvent**를 사용하여 제어합니다.

 1:  public boolean onTouchEvent(final **MotionEvent event**) {                                       
 2:     queueEvent (  
 3:              new Runnable()   
 4:             {                                                               
 5:                  public void **run**()   
 6:                  {                                                                   
 7:                     \_**renderer**.**setColor**(**event**.getX() / getWidth(), **event**.getY() / getHeight(), 1.0f);  
 8:                  }                                                                                     
 9:             }  
10:     );                                                                                       
11:     return true;                                                                              
12: }

**run**() 함수에는  **MotionEvent**의 값에 따라 색깔을 변경 할 수 있도록 

**\_renderer(VortexRenderer)** 클래스의 **setColor**()함수가 구현되어 있습니다.

첨부파일을 실행하면 다음과 같이 배경화면의 색깔이 변경되는 결과를 얻을 수 있습니다.





**[출처]** [1. 안드로이드 3D 게임 튜토리얼(구글 번역기 버전)](http://blog.naver.com/sei2001/122314871)|**작성자** [이야호](http://blog.naver.com/sei2001)