Tutorial Threejs – Parte I , Introducción

Hola a todos.

Vamos a comenzar una serie de posts dedicados a Threejs , que espero que sea de vuestro agrado.

¿Que es Threejs?

Threejs es motor 3D ligero para javascript .

Funciona muy bien bajo webGL , aprovechando la aceleración de nuestra tarjeta gráfica , dando unos resultados espectaculares.

Podemos encontrar unos ejemplos de lo que se puede hacer en la página de ejemplos oficial de Threejs.

Ésta serie de artículos pretende explicar el funcionamiento de Threejs , que hacer , como empezar , como crear escenas , importar objetos , etc , etc.

Empezaremos explicando un poco que es necesario para empezar:

Un navegador con soporte webGL , actualmente los mejores resultados los da Google Chrome , aunque se puede utilizar Mozilla Firefox , aunque es más pesado y por lo tanto un poco más lento , no olvidemos que Chrome utiliza el motor webkit que utilizan también muchos smartphones , mientras que Mozilla utiliza el motor Gecko , basado en java .

Conocimientos de programación , javascript , y html5 .

Nociones de 3D , para no perderse , o mucha paciencia para absorver conceptos nuevos.

Descargar la librería Threejs con los ejemplos y herramientas

que iremos utilizando y explicando.

Que podemos hacer !

Con Threejs podemos hacer casi cualquier cosa que podemos hacer con un motor 3D , vamos a explicarlo para que sirva también como tutorial de iniciación al 3D.

Básicamente consiste en crear una escena , con objetos dentro que visualizaremos en la pantalla , en nuestro caso en un canvas de html5 , a través de cámaras e iluminados por luces .

La escena

Es la representación del mundo virtual tridimensional , donde se irán colocando los demás elementos , es algo así como un contenedor del espacio en 3D , que empieza estando vacío .

Este espacio está vacío , no tiene nada , iremos colocando por turno los objetos que deseamos visualizar , basándonos en que la escena se pude medir , con eje de coordenadas virtual a partir del cual iremos posicionando los objetos en las 3 dimensiones básicas .

Gracias a esta referencia dentro de la escena podemos rotar los objetos , desplazarlos , etc , etc.

La escena tiene un fondo que se puede personalizar , que sería un poco así lo que sería el horizonte , se pude dejar un color plano , o una imagen , depende de lo que se quiera hacer.

La cámara

La cámara nos permite visualizar la escena desde un punto de vista concreto .
Se comporta como un objeto más , se puede rotar , desplazar , pero además tiene algunas propiedades más , propias de una cámara .

Podemos tener varias cámaras y utilizar una u otra según

convenga.

Las luces

Las luces son objetos que irradian luz y nos permiten visualizar los objetos gracias a la reflexión de la luz en ellos.

Hay varios tipos de luz , de punto (Point), solar (sun), foco (Spot) , en otras aplicaciones podemos encontrar más tipos de luz.

La intensidad y el color de las luces son configurables y nos permiten una multitud de posibilidades y pueden provocar sombras sobre los objetos de la escena.

Cada objeto reacciona diferente a la luz según sus valores para la luz de difusión (color reflejado) , especular (color refractado) y emisión (color emitido por ejemplo una bombilla) , variando según el color e intensidad.

Para una iluminación correcta es necesario el uso de normales , que son el cálculo de la perpendicular de cada cara visible del objeto .

Para empezar , la mayoría de aplicaciones comienzan con un escena que contiene una cámara y una luz , en muchos casos incluye un objeto en modo de ejemplo que suele ser un cubo o una esfera .

Los objetos

Los objetos son una representación de arrays de vértices que forman líneas y caras.
Jeje que fácil no ,

Los vértices son puntos que ocupan unas coordenadas .
La unión de 2 vértices forma líneas.
La unión de varias líneas forman planos que se suelen llamar caras.

Las caras se suelen formar por 3 o 4 líneas que le dan forma , y obviamente una cara/plano de 4 líneas se puede representar como 2 caras/planos de 3 líneas , así que lo más normal es encontrarnos con modelos de objetos formados por arrays de vértices que representan estos triángulos .

En la mayoría de aplicaciones para modelar se especifica al exportar si se quieren crear triángulos (3 líneas) en aquellas caras que son romboides (4 líneas) , para su uso posterior.

Este array de vértices que comentamos no es más que la forma del objeto , hay que posicionar el objeto en la escena .

Los objetos están hechos con uno o varios materiales a los que hay que asignarle valores según tenga que reaccionar a la luz un color de difusión , otro de especular y el color de emisión si emite luz.

Cada material puede tener una textura , que puede ser una imagen , hasta en ocasiones un vídeo.

Las texturas dan el toque de calidad a los objetos , siendo unos de los temas más importantes del desarrollo en 3d ,por ejemplo podemos aplicar transparencias gracias a las texturas , o de un plano básico aplicarle una textura para que parezca una casa.

Los objetos pueden ser de formas básicas como un cubo o una esfera , o puede ser una figura compleja , pero en el caso de ser una figura compleja mejor crearla con un editor externo (no con código me refiero) , e importarlo como un modelo.

Los modelos

Los modelos 3D son objetos pre-diseñados que se utilizan para incorporar en la escena. Pueden incluir sus propias texturas , y estar formados por uno o varios objetos , a su vez con sus propias texturas.

Los modelos se comportan como una unidad , y se ubican a

partir de un punto que es su centro de coordenadas , que se suele ubicar abajo , pero puede variar según el creador.

Podemos diseñar objetos en aplicaciones como Blender , con la comodidad que ello aporta , y exportarlos para el uso en nuestras aplicaciones , Threejs nos aporta herramientas para convertir objetos en formato obj en objetos en formato compatible para Threejs en json.

Los modelos pueden tener animaciones programadas preparadas para nuestro uso , por ejemplo ficheros md2 con acciones como saltar , correr , cargar , etc .

Threejs nos aporta una herramienta para crear modelos en formato Threejs , muy útil y práctica.

Resumiendo un poco , ya sabemos que tenemos que crear una escena , añadirle una cámara , una luz , y los objetos y modelos que queramos.

Bueno hasta aquí la introducción , en el próximo artículo pasaremos a la acción creando la primera escena .

Saludos ,
Scuraki

Tutorial Arduino parte 4

Hola a todos .

Este es el cuarto tutorial sobre Arduino , en este tutorial vamos a continuar el tutorial 3 , creando una aplicación 3d para Android , que encienda la bombilla de forma remota .

Creamos un proyecto Android nuevo en eclipse con Android SDK.

Continuando como teníamos en el tutorial 3 , en la parte del servidor php , el fichero que recibía una variable GET y encendía o apagaba la bombilla si la variable es high o low.

En este primer código podemos ver como desde el proyecto Android podemos realizar la conexión con el servidor PHP para que envíe la señal a la placa Arduino.

El fichero PHP ha de estar en modo servicio , con las lineas //die… descomentadas , para que nos devuelva una respuesta.

```
private void lighting()
{
ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
if (cm != null) {
boolean connected=false;

NetworkInfo[] info = cm.getAllNetworkInfo();
if (info != null) {
for (int i = 0; i < info.length; i++) { if (info[i].getState() == NetworkInfo.State.CONNECTED) { connected = true; } } } if(connected) { String destiny="http://192.168.0.131/arduino/service.php?check=on"; String missatxe="Signal CHECK sended. Click again to switch ON"; cambiarTextoBoton("Switch ON"); switch(estado) { case 0: destiny ="http://192.168.0.131/arduino/service.php?hight=on"; missatxe="Signal ON sended. Click again to switch OFF"; cambiarTextoBoton("Switch OFF"); break; case 1: destiny ="http://192.168.0.131/arduino/service.php?low=on"; missatxe="Signal OFF sended. Click again to switch ON"; cambiarTextoBoton("Switch ON"); break; } escribir(missatxe); InputStream is = null; //initialize String result = ""; try{ HttpClient httpclient = new DefaultHttpClient(); HttpPost httppost = new HttpPost(destiny); HttpResponse response = httpclient.execute(httppost); HttpEntity entity = response.getEntity(); is = entity.getContent(); //convert response to string try{ BufferedReader reader = new
```

```
BufferedReader(new    InputStreamReader(is,"UTF-8"),8);
StringBuilder sb = new StringBuilder(); String line = null;
while ((line = reader.readLine()) != null) { sb.append(line +
"\n");  }  is.close();  result=sb.toString();  ArrayList
message=this.parse( result);
String status=message.get(0).toString();
String signal=message.get(1).toString();
String mensage=message.get(2).toString();
if(signal.compareTo("1")==0)
{
checkOK=1;
estado=0;
}
else
{
if(signal.compareTo("2")==0)
{
checkOK=2;
estado=1;
}
else
{
if(signal.compareTo("-1")==0)
{
checkOK=-1;
estado=-1;
}
}

}
}catch(Exception e){
Log.e("log_tag", "Error converting result "+e.toString());
escribir("Error converting result "+e.toString());
}
}catch(Exception e){
Log.e("log_tag", "Error in http connection "+e.toString());
escribir("Error in http connection "+e.toString());
```

```
}
}
}

}
```

Bien , ya podemos comunicarnos con el servidor , pero bueno un poco rancia la aplicación , no?

¿Falta un botón , un poco de color o algo no?

Puesto a investigar un poco pués encontré una librería que permitía crear aplicaciones OPENGL de manera un poco más sencilla de lo normal , con un lenguage más para un principiante como yo que ir directamente a OPENGL de Android , que es un poco más rudo para alguien que no lo usa muy a menudo.

Esa librería es Min3D.

Min3D nos permite crear escenas 3D e importar objetos 3DS o OBJ , así como texturas y objetos básicos como esferas y cubos.

Tras descargar min3d de su repositorio , incluimos la carpeta al proyecto y podemos ver y probar los ejemplos.

Partiendo del ejemplo ExampleLoadObjFileMultiple.java , donde podemos ver como cargar un objeto desde un fichero .OBJ .

En este fichero podemos ver que hereda de la clase extends RendererActivity , y lo aplicaremos a la clase que estemos utilizando .

El método initScene() es el encargado de crear la escena , en este método es donde colocaremos todos los objetos de la escena , cargaremos todos los ficheros necesarios , .OBJ , .PNG , y crearemos las esferas y rectángulos que necesitemos.

Este método es llamado al iniciar la aplicación y cada vez que

el dispositivo entra en modo PAUSE , o se bloquea la pantalla.

Como en toda aplicación 3D , además de una escena necesitamos una cámara y luces para iluminar nuestros objetos.

En este método asignaremos un color de fondo , o una textura , situaremos las luces , los objetos y situaremos y enfocaremos la cámara , para obtener la perspectiva adecuada.

El método updateScene() se ejecuta en cada frame , sería el equivalente al típico evento draw() de repintado de la pantalla en aplicaciones 2D , osea que se ejecuta cada vez que se pinta la pantalla.

Éste método es el encargado de las instrucciones de las animaciones , por ejemplo si queremos rotar un objeto , en este método calculamos el valor nuevo de la rotación y se le asigna.

Para cargar los ficheros .OBJ en la aplicación Android es necesario modificar los nombres de los archivos.

Los ficheros .OBJ suelen ir acompañados de un fichero con el mismo nombre con extensión .mtl con la definición de los materiales de los grupos de los objetos que hayan definidos en el fichero .OBJ , además de las imágenes de las texturas.

Osea que el fichero .OBJ tiene objetos , valores de vértices y objetos y el fichero .mtl los materiales.

Las imágenes de las texturas las colocaremos en la carpeta res/drawable… .

Los ficheros .OBJ y .MTL los renombramos sustituyendo el punto por un «_» osea un guión bajo , para evitar los conocidos conflictos de ficheros con el mismo nombre y diferente extensión que tiene la programación con Android.

Para este tutorial hemos creado una farola , con una esfera que hace de bombilla , y una esfera con una textura con

transparencia .PNG , que hará el efecto de que la farola está encendida.

También se ha creado una especie de bullofa que emite unas esfera con transparencia , emulando el envío de ondas , dando a entender que se está comunicando con el servidor , una esfera nos indicará según la textura que tenga el texto ON de color verde , o el texto OFF de color roja o de color ambar si no hay conexión con el servidor ,a además hay otra bola que nos indicará según su textura , si estamos a más de cierta distancia de la bombilla , si hay cobertura GPS .

Bueno abrimos blender y creamos una lámpara y una bullofa (algo que de el efecto de ser un emisor de ondas) , y las exportamos en formato wavefront .OBJ .

Si no queremos utilizar blender o otro software para crear objetos 3D , los podemos descargar de alguna página , por ejemplo ésta , que ofrezcan objetos 3D en formato .OBJ .

Podemos ver como asignar el color de fondo:

```
scene.backgroundColor().setAll(0xfff2d533);
```

Asignar una textura a un rectángulo:

```
Bitmap b = Utils.makeBitmapFromResourceId(this, R.drawable.scuraki);
float w = 20f;
float h = w * (float)b.getHeight() / (float)b.getWidth();
Rectangle suelo = new Rectangle(w, h, 1,1, new Color4());
suelo.doubleSidedEnabled(true); // ... so that the back of the plane is visible
suelo.normalsEnabled(false);
scene.addChild(suelo);

Shared.textureManager().addTextureId(b, "scuraki", false);
suelo.textures().addById("scuraki");
```

Asignar una textura a una esfera:

```
b = Utils.makeBitmapFromResourceId(R.drawable.bolaroja);
```

```
Shared.textureManager().addTextureId(b, "bolaroja", false);
bolarojaTexture = new TextureVo("bolaroja");
esfera.textures().addReplace(bolarojaTexture);
```

Iluminar la escena:

```
luzobj = new Light();
luzobj.ambient.setAll(new Color4 (128,128,128,128));
luzobj.diffuse.setAll(new Color4 (64,64,164, 128));
luzobj.emissive.setAll(new Color4 (0,0,0,255));
luzobj.specular.setAll(new Color4 (0,0,0,255));
luzobj.type(LightType.POSITIONAL);
scene.lights().add(luzobj);
luzobj.position.setAll(0.65f, -0.85f, 3.5f);
```

Añadir un objeto .OBJ a la escena:

```
parser2 = Parser.createParser(Parser.Type.OBJ, getResources(),
"adictosalainformatica.min3DAdictos:raw/fanal_obj", true);

parser2.parse();

fanal = parser2.getParsedObject();
fanal.scale().y = 0.25f;
fanal.scale().z = 0.25f;
fanal.scale().x = 0.25f;
fanal.shadeModel(ShadeModel.SMOOTH);
fanal.vertexColorsEnabled(true);
fanal.normalsEnabled(true);
fanal.colorMaterialEnabled(false);

scene.addChild(fanal);
fanal.position().x=0.0f;
fanal.position().y=1.6f;
fanal.position().z=-5f;
fanal.rotation().x=25f;
```

Controlar la rotación de los objetos en el método updateScene:

```
bombilla.rotation().y=yrot;
bombilla.rotation().x=xrot;
receptor.rotation().y=yrot;
```

```
receptor.rotation().x=xrot;

xrot += xspeed;
yrot += yspeed;
_count++;
```

Para dar un toque de color la bullofa cambia el valor de escalado cada cierto tiempo , y se crean unas esferas con transparencia , que viajan desde la bullofa (que está cerca de la cámara) hasta la lámpara (que está al fondo de la escena), emulando el envio de ondas.

Un array de esferas ameniza la pantalla , desplazandose .

La clase Burbuja es la encargada de controlar estas esferas:

```
package adictosalainformatica.min3DAdictos;

import android.graphics.Bitmap;
import min3d.Shared;
import min3d.Utils;
import min3d.core.Scene;
import min3d.objectPrimitives.Sphere;
import min3d.vos.Color4;
import min3d.vos.TextureVo;
public class Burbuja {
private float velocidad_x=.000f;
private float velocidad_y=0.02f;
private float velocidad_z=0.04f;
private float posicion_x=0.45f;
private float posicion_y=-0.25f;
private float posicion_z=0.3f;
private float variacion_x=0.45f;
private float variacion_y=-0.25f;
private float variacion_z=0.6f;
private float limite_x=4.01f;
private float limite_y=4.01f;
private float limite_z=4.8f;
private Color4 color=null;
```

```
private Sphere esfera =null;
long _index;
private float contador=0;
public Burbuja(long index)
{
this._index=index;
}
public void make(Scene scene )
{
if(esfera!=null)
{
esfera.clear();
}
esfera=null;
TextureVo textura=null;
esfera = new Sphere(1.5f, 20,20);
esfera.scale().x = esfera.scale().y = esfera.scale().z = .1f;
esfera.position().setAll(posicion_x, posicion_y, posicion_z);
Bitmap                              b                                  =
Utils.makeBitmapFromResourceId(R.drawable.tscuraki);
Shared.textureManager().addTextureId(b, "burbuja0" + _index,
false);
b.recycle();
textura = new TextureVo("burbuja0" + _index);
esfera.textures().addReplace(textura);
esfera.colorMaterialEnabled(false);
esfera.vertexColorsEnabled(false);
esfera.lightingEnabled();
scene.addChild(esfera);

//Log.v(Min3d.TAG, "ReCrea Burbuja=" + _index);

}

public int mover()
{
int moviendo=1;
posicion_x=variacion_x + ( contador * velocidad_x);
```

```
posicion_y=variacion_y + ( contador * velocidad_y);
posicion_z=variacion_z - ( contador * velocidad_z);
esfera.position().setAll(posicion_x, posicion_y, posicion_z);
if((posicion_x>limite_x)||(posicion_y>limite_y)||(posicion_z *
-1 > limite_z ))
{
esfera.clear();

moviendo=0;
}
//Log.v(Min3d.TAG, "Mueve Burbuja=" + _index + " posiciónx="+
posicion_x + " posicióny=" + posicion_y + " posiciónz=" +
posicion_z);
esfera.rotation().x=contador * -1.3f ;
esfera.rotation().y=contador * -1.3f;
esfera.rotation().z=contador * -1.3f;
esfera.scale().x = esfera.scale().y = esfera.scale().z = .1f +
(contador * 0.001f);
contador++;
return moviendo;

}

public float getLimite_x() {
return limite_x;
}
public void setLimite_x(float limite_x) {
this.limite_x = limite_x;
}
public float getLimite_y() {
return limite_y;
}
public void setLimite_y(float limite_y) {
this.limite_y = limite_y;
}
public float getLimite_z() {
return limite_z;
}
```

```
public void setLimite_z(float limite_z) {
this.limite_z = limite_z;
}
public Color4 getColor() {
return color;
}
public void setColor(Color4 color) {
this.color = color;
}
public float getVelocidad_x() {
return velocidad_x;
}
public void setVelocidad_x(float velocidad_x) {
this.velocidad_x = velocidad_x;
}
public float getVelocidad_y() {
return velocidad_y;
}
public void setVelocidad_y(float velocidad_y) {
this.velocidad_y = velocidad_y;
}
public float getVelocidad_z() {
return velocidad_z;
}
public void setVelocidad_z(float velocidad_z) {
this.velocidad_z = velocidad_z;
}
public float getPosicion_x() {
return posicion_x;
}
public void setPosicion_x(float posicion_x) {
this.posicion_x = posicion_x;
}
public float getPosicion_y() {
return posicion_y;
}
public void setPosicion_y(float posicion_y) {
```

```
this.posicion_y = posicion_y;
}
public float getPosicion_z() {
return posicion_z;
}
public void setPosicion_z(float posicion_z) {
this.posicion_z = posicion_z;
}
public float getVariacion_x() {
return variacion_x;
}
public void setVariacion_x(float variacion_x) {
this.variacion_x = variacion_x;
}
public float getVariacion_y() {
return variacion_y;
}
public void setVariacion_y(float variacion_y) {
this.variacion_y = variacion_y;
}
public float getVariacion_z() {
return variacion_z;
}
public void setVariacion_z(float variacion_z) {
this.variacion_z = variacion_z;
}
public Sphere getEsfera() {
return esfera;
}
public void setEsfera(Sphere esfera) {
this.esfera = esfera;
}
public long getIndex() {
return _index;
}
public void setIndex(long _index) {
this._index = _index;
```

```
}

}
```

Finalmente controlamos la distancia a la bombilla gracias al sensor GPS , activándolo si es necesario.

Podemos decidir activar la bombilla a cierta distancia , útil para luces de garages , por ejemplo que se encienda cuando falta 2 kilómetros para llegar con el coche.

En nuestro caso tenemos la esfera que nos indica la distancia , modificamos la textura para que nos muestre FAR si está lejos y NEAR si está cerca.

```
private void loadJipiEs()
{
// Acquire a reference to the system Location Manager
this.locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

if
(!this.locationManager.isProviderEnabled(LocationManager.GPS_P
ROVIDER))
{
createGpsDisabledAlert();
}
else
{
// List all providers:
List providers = this.locationManager.getAllProviders();

Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);

this.bestProvider =
this.locationManager.getBestProvider(criteria, false);

Location mylocation =
```

```
this.locationManager.getLastKnownLocation(this.bestProvider);

if(mylocation!=null)
{
this.getLocation(mylocation);
}
else
{
//this.afegir("Posición inicial vacía.");
}
}

}
private void getLocation(Location location)
{
if(location!=null)
{

boolean hasAltitude=false;
boolean hasAccuracy=false;
boolean hasBearing=false;
lat=location.getLatitude();
lon=location.getLongitude();
alt=location.getAltitude();

hasAltitude=location.hasAltitude();
hasAccuracy=location.hasAccuracy();
hasBearing=location.hasBearing();

distance=location.distanceTo(coords);
if(distance<5000) { distanceState="near"; } else {
distanceState="far"; } } else { distanceState="Unknown"; }
//Toast.makeText(getApplicationContext(), "Distance is : " +
distance, Toast.LENGTH_LONG).show(); /*runOnUiThread(new
Runnable() { public void run() {
Toast.makeText(getApplicationContext(), "Distance is : " +
distance, Toast.LENGTH_LONG).show();
missatger.setText("Distance is : " + distance); } });*/
```

```
escribir("Distance is : " + distance); } private void createGpsDisabledAlert(){ AlertDialog.Builder builder = new AlertDialog.Builder(this); builder.setMessage("Your GPS is disabled! Would you like to enable it?") .setCancelable(false) .setPositiveButton("Enable GPS", new DialogInterface.OnClickListener(){ public void onClick(DialogInterface dialog, int id){ showGpsOptions(); } }); builder.setNegativeButton("Do nothing", new DialogInterface.OnClickListener(){ public void onClick(DialogInterface dialog, int id){ dialog.cancel(); } }); AlertDialog alert = builder.create(); alert.show(); } private void showGpsOptions(){ Intent gpsOptionsIntent = new Intent( android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS); startActivity(gpsOptionsIntent); } }
```

En nuestro caso la luz la hemos activado a través de un botón externo a la escena , esta señal modifica la textura de la esfera que nos indica el estado de la bombilla , según sea la respuesta del servidor , ON , OFF o sin conexión , si la bombila está ON la esfera que emula la bombilla encendida , que está en la lámpara , se hace visible rodeando una esfera más pequeña que hace de núcleo de la bombilla.

Tutorial Arduino parte 3

En este tutorial veremos como conectar la placa Arduino a la red de 220V de nuestra casa , y poder utilizarlo para controlar nuestros aparatos eléctricos .

En este ejemplo mostraremos como encender una bombilla a 220V desde un lugar remoto .

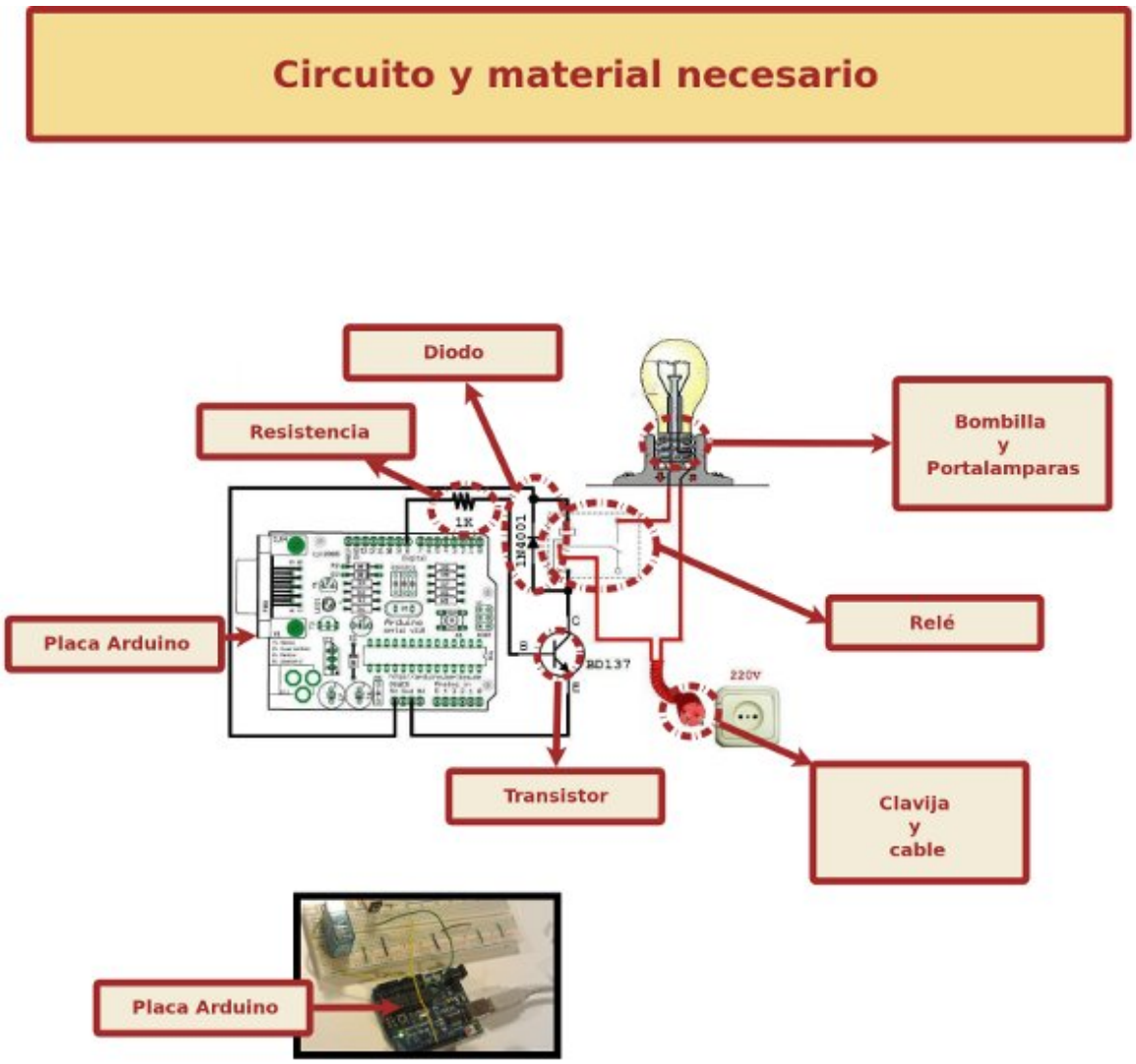
Este tutorial está basado en un tutorial de la página de

Arduino , ver referencias al final del post.

En el próximo tutorial explicaremos como utilizar una aplicación android en 3D para crear un interfaz para este tutorial , que podremos utilizar en nuestro smartphone Android.

Material necesario:

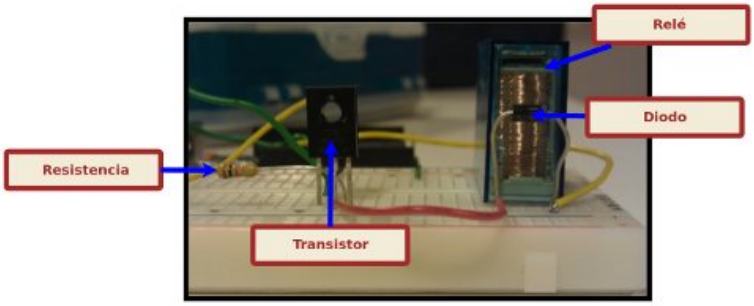
-Placa Arduino
-1 Relé de 5V ó 6V a 220V
-1 Diodo
-1 Transistor 5V
-1 Resistencia
-Cable
-1 Clavija
-1 Portalamparas
-1 Bombilla

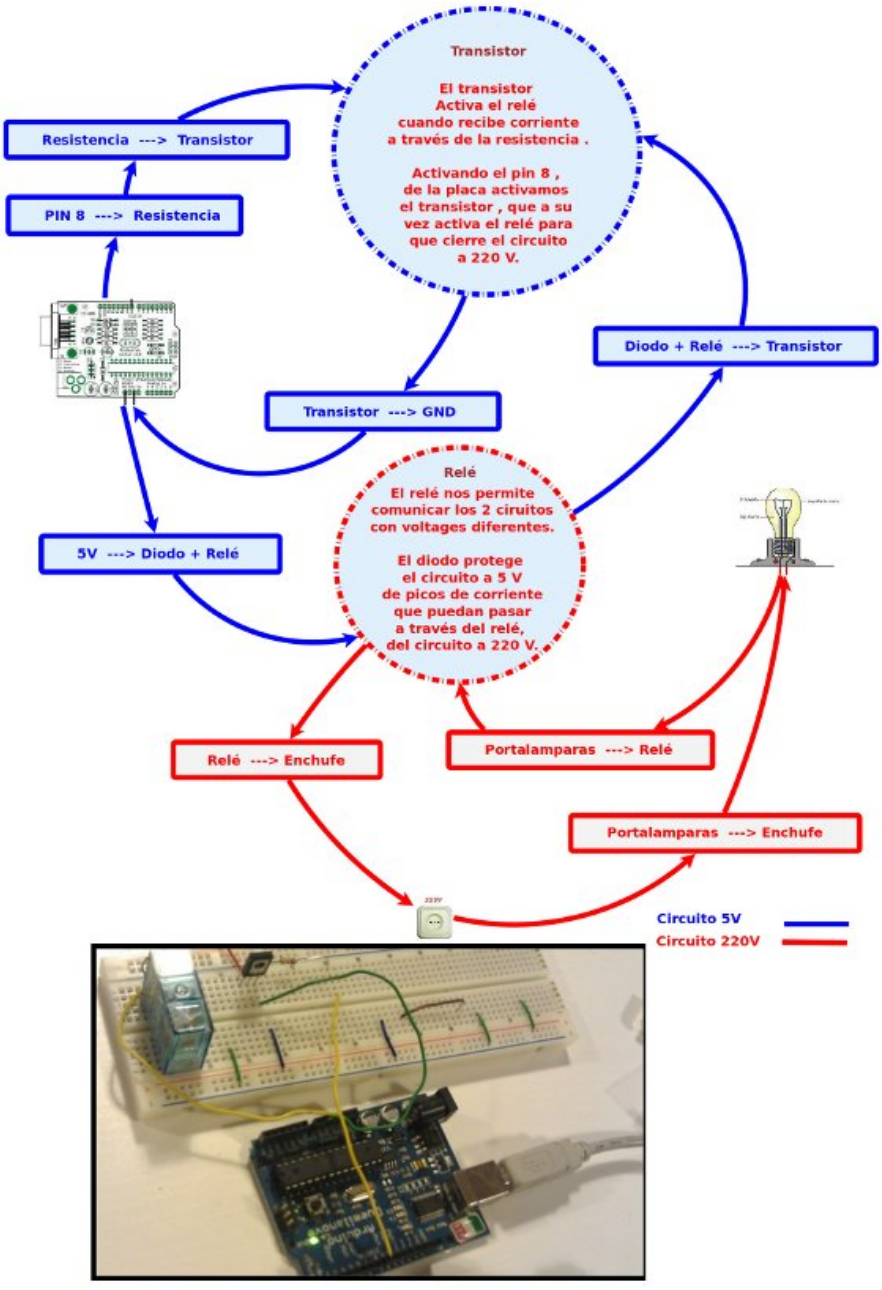


Material necesario

Una vez tengamos el material necesario procedemos a realizar el cableado de los circuitos .

Descripción del cableado de los circuitos





descripción circuitos

Una vez preparado el cableado , el código necesario para la placa Arduino es muy sencillo.
Utilizamos el pin 8 para activar el relé por medio del transistor.

Abrimos el puerto USB 115200 en modo lectura para recibir las notificaciones que nos enviará el servidor PHP.

Si por el puerto nos entra el caracter 2 , activa el relé , si es 1 lo desactiva.

Código Arduino:

```
int ledPin = 8;

int number_in = 0;

void setup() {

pinMode(ledPin, OUTPUT);

Serial.begin(115200);

}

void loop() {

if (Serial.available() > 0) {

number_in = Serial.read();

}

if (number_in > 0) {

if(number_in==2)
{
digitalWrite(ledPin, HIGH);
}
else
{
```

```
if(number_in==1)
{
digitalWrite(ledPin, LOW);
}
}

}

number_in = 0;

}
```

Utilizaremos un simple servidor php para enviar la señal a la placa Arduino conectada a nuestro servidor GNU/linux , en este caso a través del cable USB.

En nuestro caso la ruta al dispositivo que vamos a utilizar es /dev/ttyUSB0.

El puerto de la conexión USB que utilizaremos para comunicarnos con la placa Arduino será el 115200 .

Para encender o apagar la bombilla enviaremos por get a la ruta de nuestro servidor la variable hight , para encender , y la variable low para apagar.

Finalmente el servidor responde con un mensaje si ha recibido una señal correcta.

Código PHP:

```
<?php // Nonzero number to be sent to Arduino $c = 0; if(isset($_GET["hight"])) { $c=2; } if(isset($_GET["low"])) { $c=1; } if($c>0)
{
// Include the PHP serial class
require_once("phpSerial.php");

// Start a new serial class

$serial = new phpSerial;
```

```
// Specify the device being used

$serial->deviceSet("/dev/ttyUSB0");

// Set baud rate

$serial->confBaudRate(115200);

$serial->confParity("none");

$serial->confCharacterLength(8);

$serial->confStopBits(1);

$serial->confFlowControl("none");

// Open the device

$serial->deviceOpen();

// Write to the device

$serial->sendMessage(chr($c));

// Close the port

$serial->deviceClose();
$message= "Signal ".$c." Received! .
";

die(json_encode(array('status' => 'success', 'data' =>
$message)));
}
else
{
$message= "No Signal Received! .";
die(json_encode(array('status' => 'failed', 'data' =>
$message)));
}
?>
```

Y ya lo tenemos , tenemos una página php que nos funciona como

un servicio para poder encender o apagar una bombilla .

Podríamos crear en este punto una pequeña interfaz html , aprovechando el mismo fichero .php , comentando las líneas die(… , creando unos botones para encender y apagar la bombilla , pero nuestra intención es utilizarlo como servicio para una aplicación Android que os explicaremos en el próximo tutorial.

En este ejemplo hemos utilizado la librería phpSerial.php ,no recuerdo de donde la saqué así que os dejo el código , aunque en principio cualquier librería que os permita comunicar con el puerto serie debería valer.

este es el código:

```
<?php define ("SERIAL_DEVICE_NOTSET", 0); define ("SERIAL_DEVICE_SET", 1); define ("SERIAL_DEVICE_OPENED", 2); /** * Serial port control class * * THIS PROGRAM COMES WITH ABSOLUTELY NO WARANTIES ! * USE IT AT YOUR OWN RISKS ! * * @author Rémy Sanchez  * @thanks Aurélien Derouineau for finding how to open serial ports with windows
* @thanks Alec Avedisyan for help and testing with reading
* @copyright under GPL 2 licence
*/
class phpSerial
{
var $_device = null;
var $_windevice = null;
var $_dHandle = null;
var $_dState = SERIAL_DEVICE_NOTSET;
var $_buffer = "";
var $_os = "";

/**
* This var says if buffer should be flushed by sendMessage (true) or manualy (false)
*
```

```
* @var bool
*/
var $autoflush = true;

/**
* Constructor. Perform some checks about the OS and setserial
*
* @return phpSerial
*/
function phpSerial ()
{
setlocale(LC_ALL, "en_US");

$sysname = php_uname();

if (substr($sysname, 0, 5) === "Linux")
{
$this->_os = "linux";

if($this->_exec("stty --version") === 0)
{
register_shutdown_function(array($this, "deviceClose"));
}
else
{
trigger_error("No  stty  availible,  unable  to  run.",
E_USER_ERROR);
}
}
elseif(substr($sysname, 0, 7) === "Windows")
{
$this->_os = "windows";
register_shutdown_function(array($this, "deviceClose"));
}
else
{
trigger_error("Host OS is neither linux nor windows, unable tu
run.", E_USER_ERROR);
```

```
exit();
}
}

//
// OPEN/CLOSE DEVICE SECTION -- {START}
//

/**
* Device set function : used to set the device name/address.
* -> linux : use the device address, like /dev/ttyS0
* -> windows : use the COMxx device name, like COM1 (can also
be used
* with linux)
*
* @param string $device the name of the device to be used
* @return bool
*/
function deviceSet ($device)
{
if ($this->_dState !== SERIAL_DEVICE_OPENED)
{
if ($this->_os === "linux")
{
if (preg_match("@^COM(\d+):?$@i", $device, $matches))
{
$device = "/dev/ttyS" . ($matches[1] - 1);
}

if ($this->_exec("stty -F " . $device) === 0)
{
$this->_device = $device;
$this->_dState = SERIAL_DEVICE_SET;
return true;
}
}
elseif ($this->_os === "windows")
{
```

```
if (preg_match("@^COM(\d+):?$@i", $device, $matches) and
$this->_exec(exec("mode " . $device)) === 0)
{
$this->_windevice = "COM" . $matches[1];
$this->_device = "\\.\com" . $matches[1];
$this->_dState = SERIAL_DEVICE_SET;
return true;
}
}

trigger_error("Specified serial port is not valid",
E_USER_WARNING);
return false;
}
else
{
trigger_error("You must close your device before to set an
other one", E_USER_WARNING);
return false;
}
}

/**
* Opens the device for reading and/or writing.
*
* @param string $mode Opening mode : same parameter as fopen()
* @return bool
*/
function deviceOpen ($mode = "r+b")
{
if ($this->_dState === SERIAL_DEVICE_OPENED)
{
trigger_error("The device is already opened", E_USER_NOTICE);
return true;
}

if ($this->_dState === SERIAL_DEVICE_NOTSET)
{
```

```
trigger_error("The device must be set before to be open",
E_USER_WARNING);
return false;
}

if (!preg_match("@^[raw]\+?b?$@", $mode))
{
trigger_error("Invalid opening mode : ".$mode.". Use fopen()
modes.", E_USER_WARNING);
return false;
}

$this->_dHandle = @fopen($this->_device, $mode);

if ($this->_dHandle !== false)
{
stream_set_blocking($this->_dHandle, 0);
$this->_dState = SERIAL_DEVICE_OPENED;
return true;
}

$this->_dHandle = null;
trigger_error("Unable to open the device", E_USER_WARNING);
return false;
}

/**
* Closes the device
*
* @return bool
*/
function deviceClose ()
{
if ($this->_dState !== SERIAL_DEVICE_OPENED)
{
return true;
}

if (fclose($this->_dHandle))
```

```
{
$this->_dHandle = null;
$this->_dState = SERIAL_DEVICE_SET;
return true;
}

trigger_error("Unable to close the device", E_USER_ERROR);
return false;
}

//
// OPEN/CLOSE DEVICE SECTION -- {STOP}
//

//
// CONFIGURE SECTION -- {START}
//

/**
* Configure the Baud Rate
* Possible rates : 110, 150, 300, 600, 1200, 2400, 4800, 9600,
38400,
* 57600 and 115200.
*
* @param int $rate the rate to set the port in
* @return bool
*/
function confBaudRate ($rate)
{
if ($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set the baud rate : the device is
either not set or opened", E_USER_WARNING);
return false;
}

$validBauds = array (
110 => 11,
```

```
150 => 15,
300 => 30,
600 => 60,
1200 => 12,
2400 => 24,
4800 => 48,
9600 => 96,
19200 => 19,
38400 => 38400,
57600 => 57600,
115200 => 115200
);

if (isset($validBauds[$rate]))
{
if ($this->_os === "linux")
{
$ret = $this->_exec("stty -F " . $this->_device . " " . (int)
$rate, $out);
}
elseif ($this->_os === "windows")
{
$ret = $this->_exec("mode " . $this->_windevice . " BAUD=" .
$validBauds[$rate], $out);
}
else return false;

if ($ret !== 0)
{
trigger_error ("Unable to set baud rate: " . $out[1],
E_USER_WARNING);
return false;
}
}
}

/**
* Configure parity.
```

```
* Modes : odd, even, none
*
* @param string $parity one of the modes
* @return bool
*/
function confParity ($parity)
{
if ($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set parity : the device is either not
set or opened", E_USER_WARNING);
return false;
}

$args = array(
"none" => "-parenb",
"odd" => "parenb parodd",
"even" => "parenb -parodd",
);

if (!isset($args[$parity]))
{
trigger_error("Parity mode not supported", E_USER_WARNING);
return false;
}

if ($this->_os === "linux")
{
$ret = $this->_exec("stty -F " . $this->_device . " " .
$args[$parity], $out);
}
else
{
$ret = $this->_exec("mode " . $this->_windevice . " PARITY=" .
$parity{0}, $out);
}

if ($ret === 0)
```

```
{
return true;
}

trigger_error("Unable to set parity : " . $out[1],
E_USER_WARNING);
return false;
}

/**
* Sets the length of a character.
*
* @param int $int length of a character (5 <= length <= 8) *
@return bool */ function confCharacterLength ($int) { if
($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set length of a character : the
device is either not set or opened", E_USER_WARNING);
return false;
}

$int = (int) $int;
if ($int < 5) $int = 5; elseif ($int > 8) $int = 8;

if ($this->_os === "linux")
{
$ret = $this->_exec("stty -F " . $this->_device . " cs" .
$int, $out);
}
else
{
$ret = $this->_exec("mode " . $this->_windevice . " DATA=" .
$int, $out);
}

if ($ret === 0)
{
return true;
```

```
}

trigger_error("Unable to set character length : " .$out[1],
E_USER_WARNING);
return false;
}

/**
* Sets the length of stop bits.
*
* @param float $length the length of a stop bit. It must be
either 1,
* 1.5 or 2. 1.5 is not supported under linux and on some
computers.
* @return bool
*/
function confStopBits ($length)
{
if ($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set the length of a stop bit : the
device is either not set or opened", E_USER_WARNING);
return false;
}

if ($length != 1 and $length != 2 and $length != 1.5 and
!($length == 1.5 and $this->_os === "linux"))
{
trigger_error("Specified stop bit length is invalid",
E_USER_WARNING);
return false;
}

if ($this->_os === "linux")
{
$ret = $this->_exec("stty -F " . $this->_device . " " .
(($length == 1) ? "-" : "") . "cstopb", $out);
}
```

```
else
{
$ret = $this->_exec("mode " . $this->_windevice . " STOP=" .
$length, $out);
}

if ($ret === 0)
{
return true;
}

trigger_error("Unable to set stop bit length : " . $out[1],
E_USER_WARNING);
return false;
}

/**
* Configures the flow control
*
* @param string $mode Set the flow control mode. Availible
modes :
* -> "none" : no flow control
* -> "rts/cts" : use RTS/CTS handshaking
* -> "xon/xoff" : use XON/XOFF protocol
* @return bool
*/
function confFlowControl ($mode)
{
if ($this->_dState !== SERIAL_DEVICE_SET)
{
trigger_error("Unable to set flow control mode : the device is
either not set or opened", E_USER_WARNING);
return false;
}

$linuxModes = array(
"none" => "clocal -crtscts -ixon -ixoff",
"rts/cts" => "-clocal crtscts -ixon -ixoff",
```

```
"xon/xoff" => "-clocal -crtscts ixon ixoff"
);
$windowsModes = array(
"none" => "xon=off octs=off rts=on",
"rts/cts" => "xon=off octs=on rts=hs",
"xon/xoff" => "xon=on octs=off rts=on",
);

if ($mode !== "none" and $mode !== "rts/cts" and $mode !==
"xon/xoff") {
trigger_error("Invalid  flow  control  mode  specified",
E_USER_ERROR);
return false;
}

if ($this->_os === "linux")
$ret = $this->_exec("stty -F " . $this->_device . " " .
$linuxModes[$mode], $out);
else
$ret = $this->_exec("mode " . $this->_windevice . " " .
$windowsModes[$mode], $out);

if ($ret === 0) return true;
else {
trigger_error("Unable to set flow control : " . $out[1],
E_USER_ERROR);
return false;
}
}

/**
* Sets a setserial parameter (cf man setserial)
* NO MORE USEFUL !
* -> No longer supported
* -> Only use it if you need it
*
* @param string $param parameter name
* @param string $arg parameter value
```

```
* @return bool
*/
function setSetserialFlag ($param, $arg = "")
{
if (!$this->_ckOpened()) return false;

$return = exec ("setserial " . $this->_device . " " . $param .
" " . $arg . " 2>&1");

if ($return{0} === "I")
{
trigger_error("setserial: Invalid flag", E_USER_WARNING);
return false;
}
elseif ($return{0} === "/")
{
trigger_error("setserial:  Error  with  device  file",
E_USER_WARNING);
return false;
}
else
{
return true;
}
}

//
// CONFIGURE SECTION -- {STOP}
//

//
// I/O SECTION -- {START}
//

/**
* Sends a string to the device
*
* @param string $str string to be sent to the device
```

```
* @param float $waitForReply time to wait for the reply (in
seconds)
*/
function sendMessage ($str, $waitForReply = 0.1)
{
$this->_buffer .= $str;

if ($this->autoflush === true) $this->flush();

usleep((int) ($waitForReply * 1000000));
}

/**
* Reads the port until no new datas are availible, then return
the content.
*
* @pararm int $count number of characters to be read (will
stop before
* if less characters are in the buffer)
* @return string
*/
function readPort ($count = 0)
{
if ($this->_dState !== SERIAL_DEVICE_OPENED)
{
trigger_error("Device must be opened to read it",
E_USER_WARNING);
return false;
}

if ($this->_os === "linux")
{
$content = ""; $i = 0;

if ($count !== 0)
{
do {
if ($i > $count) $content .= fread($this->_dHandle, ($count -
```

```
$i));
else $content .= fread($this->_dHandle, 128);
} while (($i += 128) === strlen($content));
}
else
{
do {
$content .= fread($this->_dHandle, 128);
} while (($i += 128) === strlen($content));
}

return $content;
}
elseif ($this->_os === "windows")
{
/* Do nohting : not implented yet */
}

trigger_error("Reading serial port is not implemented for
Windows", E_USER_WARNING);
return false;
}

/**
* Flushes the output buffer
*
* @return bool
*/
function flush ()
{
if (!$this->_ckOpened()) return false;

if (fwrite($this->_dHandle, $this->_buffer) !== false)
{
$this->_buffer = "";
return true;
}
else
```

```
{
$this->_buffer = "";
trigger_error("Error while sending message", E_USER_WARNING);
return false;
}
}

//
// I/O SECTION -- {STOP}
//

//
// INTERNAL TOOLKIT -- {START}
//

function _ckOpened()
{
if ($this->_dState !== SERIAL_DEVICE_OPENED)
{
trigger_error("Device must be opened", E_USER_WARNING);
return false;
}

return true;
}

function _ckClosed()
{
if ($this->_dState !== SERIAL_DEVICE_CLOSED)
{
trigger_error("Device must be closed", E_USER_WARNING);
return false;
}

return true;
}

function _exec($cmd, &$out = null)
{
```

```
$desc = array(
1 => array("pipe", "w"),
2 => array("pipe", "w")
);

$proc = proc_open($cmd, $desc, $pipes);

$ret = stream_get_contents($pipes[1]);
$err = stream_get_contents($pipes[2]);

fclose($pipes[1]);
fclose($pipes[2]);

$retVal = proc_close($proc);

if (func_num_args() == 2) $out = array($ret, $err);
return $retVal;
}

//
// INTERNAL TOOLKIT -- {STOP}
//
}
?>
```

Referencias:

[Tutorial de la página de Arduino]

[Vídeo del turorial]

Próximamente:

Tutorial Arduino parte 4

En el próximo tutorial explicaremos como utilizar una aplicación android en 3D para crear un interfaz para este tutorial , que podremos utilizar en nuestro celular Android.

Android sdk en Ubuntu unknown device ???????????

Tras instalar android sdk , eclipse , actualizar etc , nos podemos encontrar con que el emulador funciona,
pero no podemos debugar en el dispositivo.

Debugar en el dispositivo , nos ayuda a crear aplicaciones para los sensores , y nos permite probarlo al mismo tiempo , como los sensores de nivel , gps , etc , sin el dispositivo sería muy difícil .

Problema en DDMS , dispositivo desconocido ???????????
No funciona el debug con el dispositivo.

/android-sdk/platform-tools$ ./adb devices devuelve :

List of devices attached
??????????? ??
unknown device ???????????
[Fuente de la solución]
En esta zona podemos ver la solución:

> *14:25 W/ddms: Unable to get frame buffer: device (???????????) request*
> *rejected: insufficient permissions for device*
> ==
>
> *I then checked*
>
> ==
> *$ ./adb devices*
> *List of devices attached*
> *??????????? no permissions*

> ==
>
> There appears to be a permission problem. I then wrote a
> `51-android.rules' file (with chmod a+r) in /etc/udev/rules.d/, such that
>
> ==
> # cat 51-android.rules
> SUBSYSTEM==»usb», ATTRS{idVendor}==»0bb4″, ATTRS{idProduct}==»0c87″,
> MODE=»0666″
> ==
>
> (the idVendor is the HTC one). I then restarted udev, but to no avail:
> same problems. Any idea?
>
>
>

–
Merciadri Luca
See http://www.student.montefiore.ulg.ac.be/~merciadri/
I use PGP. If there is an incompatibility problem with your mail
client, please contact me.

Old 08-13-2010, 02:33 PM
Merciadri Luca

Default Android phone is not recognized by Android SDK, despite udev conf
Problem solved: restarting udev from CLI was not sufficient. I needed to
restart.

Resumiendo:

1:)-> crear el fichero /etc/udev/rules.d/51-android.rules :
$ sudo gedit /etc/udev/rules.d/51-android.rules
con el siguiente contenido:
SUBSYSTEM=="usb", ATTRS{idVendor}=="0bb4", ATTRS{idProduct}=="0c87", MODE="0666"

2:)-> darle permisos $ sudo chmod a+r /etc/udev/rules.d/51-android.rules
3:)-> reiniciar

Bueno , ya podemos utilizar el dispositivo , en la carpeta donde esté instalado adb ,
./adb devices , devuelve en mi caso

List of devices attached
HT0C4RX21596 device

ya podemos debugar , probamos creando un proyecto Android en eclipse , pulsamos el botón de debug y ya tenemos nuestro hello world …. que tanto deseábamos.

Introducción a los mundos virtuales

Los Mundos Virtuales son mundos alternativos , como ya sabemos , pero sabemos muy bien que hacer con ellos ¿no? , la mayoría son juegos como Los Sims , como World of Warcraft , en los que pasamos el tiempo libre , horas y horas , pasando miles de aventuras .

Tras la aparición de Software como Second Life , los mundos virtuales nos permiten más cosas , NO SON JUEGOS , nos

permiten crear objetos (no solo algunos predeterminados) , socializarnos por chat , web , crear y utilizar espacios para difundir información a visitantes desconocidos , nos permiten crear vídeos , aplicaciones en un espacio en 3d con cierta agilidad con lenguajes como LSL.

Es muy utilizado por ejemplo para conferencias virtuales , haciendo un streaming de vídeo , o en materias de educación para amenizar los estudios de los jóvenes , haciendo por ejemplo clases virtuales de física , entre otras materias.

¿Como empezar , como iniciarse? , Pues la mejor manera es utilizar un mundo virtual ya existente como usuario , crear un avatar , y probarlo.

¿Que es un avatar? , Pues el avatar representa a nuestro usuario dentro del mundo , normalmente podemos definir algunos aspectos de su sexo ,ropas , color , etc , suele estar representado como un humanoide , pero puede ser un objeto.
Con este avatar nos desplazamos por el mundo en 3d , donde podemos andar , correr , saltar ,bailar, volar , y la cámara de nuestro programa seguirá sus movimientos a una distancia prudencial.

Normalmente al entrar en un mundo virtual por primera vez , se aparece en una especie de recepción , que nos da un poco la bienvenida hasta que definimos una destino y ya nos disgregamos por el mundo virtual.

No está de más recordar que según donde entréis pueden haber contenidos de pago.

Second Life por ejemplo utiliza su propia moneda y para conseguirla se ha de pagar con dinero real.
También se puede conseguir de los demás , si eres bueno puedes crear objetos y llegar a venderlos , por ejemplo ropa ,pulseras , vehículos , etc.

Bueno , antes de empezar con los objetos , empezaremos por explicar que nos podemos encontrar , donde podemos ir , etc.

Normalmente el programa que utilizamos para conectar al mundo , incluye un buscador , con un mapa.
Si buscamos por una ciudad , empresa , cualquier cosa , nos devolverá un listado con los resultados de la búsqueda , y nos permitirá trasportarnos al lugar elegido.
En el mapa podemos ver un cuadrado , que representa el mar con unos puntos , que nos informan de donde están y su nombre.
También podemos apreciar unas coordenadas que podemos utilizar para transportarnos.

Una vez que llegamos a nuestro destino , no suele haber 2 iguales , aquí ya va la imaginación del propietario del lugar , o de sus ayudantes.

En un mundo virtual los contenidos los crea alguien por supuesto , en Second Life se pueden ver muchos ejemplos , pero de donde sale? , pues para que exista un mundo tiene que haber un servidor por lo menos , que contenga la información de nuestro avatar y nos permita acceder al contenido de forma virtual.

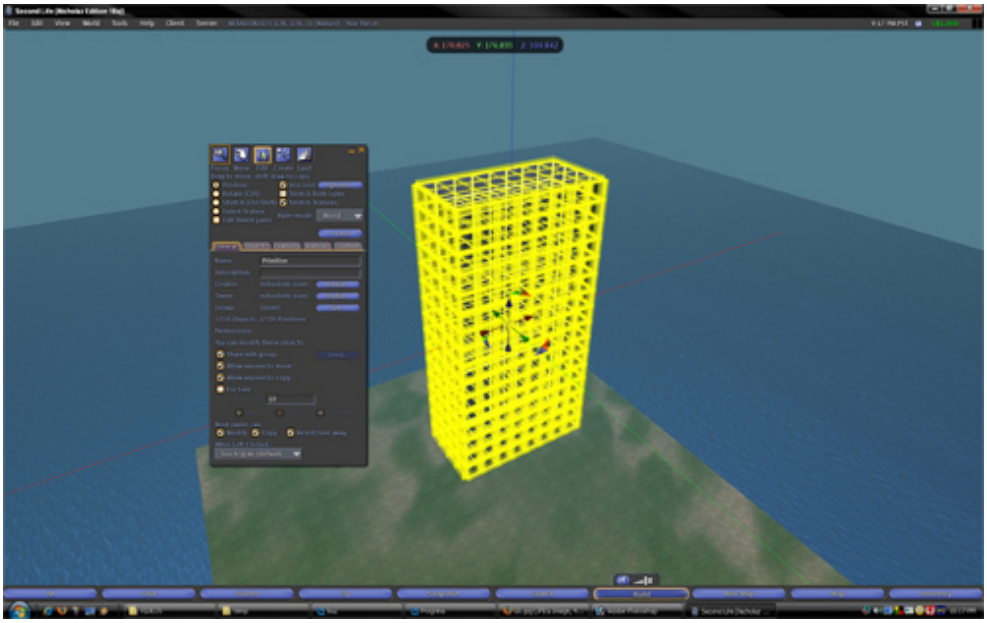
En la web de Second Life podeis consultar las tarifas de lo que cuesta una parcela de tierra.
Podemos instalar nuestro propio mundo virtual con OpenSim , donde podemos crear a nuestro antojo , sin coste alguno , o utilizar servidores gratuitos que hay unos cuantos , ya haré un post explicándolo.

Bueno las parcelas tienen un propietario , eso lo hemos dejado claro , este propietario , puede crear objetos de todo tipo ,

vehículos , ropas , etc , los cuales podemos ver , tocar , algunos los podemos copiar , vender , etc .

¿Y los demás, no pueden crear objetos? , Existen algunas zonas normalmente llamadas Sandbox , donde el propietario permite a otros usuarios crear sus propios objetos , si buscáis sandbox en el buscador seguramente encontréis lugares donde poder construir.

Los objetos que creéis son de vuestra propiedad , podéis copiarlos , programarlos , compartirlos , venderlos , en las propiedades del objeto , podéis asignarles los permisos que deseéis.



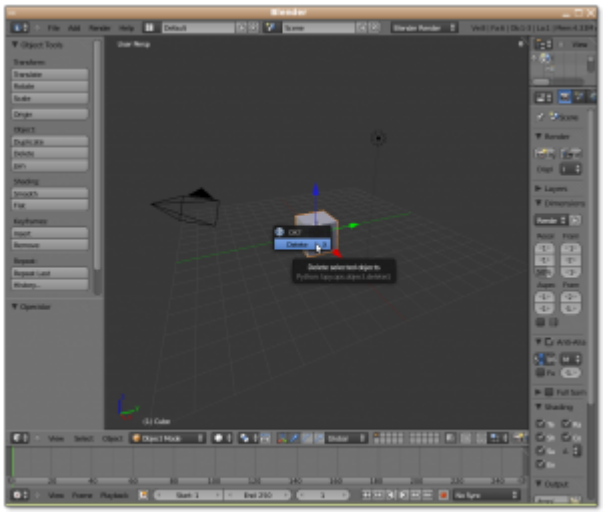
Construyendo

Del mismo modo podemos asignar permisos a nuestros scripts de programación.

Bueno hasta aquí la introducción a los mundos virtuales , más adelante explicaremos diferentes servidores , clientes a utilizar para conectarse , como crear objetos con blender y pasarlos al mundo ,exportarlos , etc.

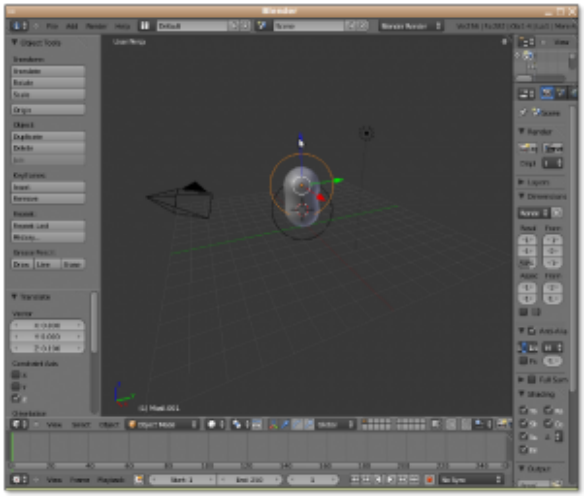
Pelo en blender Primer tutorial

Primer tutorial de blender Antes de nada Blender es un programa de modelado , animación , etc en 3D. Blender es **gratuito** , **opensource** . Para descargar Blender , visitad la página de descargas de Blender. En este tutorial se utilizará la versión **2.5** Beta para Linux. Empezaremos con una introducción a las teclas y al entorno. Al iniciar Blender nos sale por defecto un cubo , una luz y una cámara. El objeto seleccionado es el cubo , que está iluminado. Si pulsamos las teclas Ctrl + Alt y movemos la rueda del ratón veremos que la vista gira alrededor del cubo . Si pulsamos Shift y la rueda del ratón se mueve la vista en sentido vertical . Si pulsamos Ctrl y la rueda del ratón se mueve la vista en sentido horizontal. Si pulsamos con botón derecho en un objeto , se



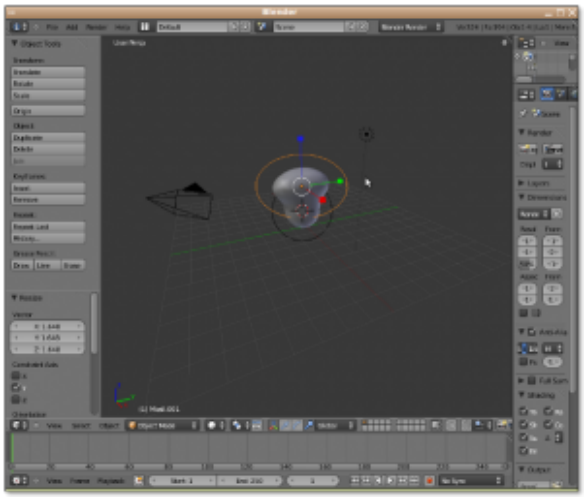
selecciona el objeto pulsado. En este enlace: http://blender.gulo.org/ encontrareis tutoriales

para iniciarse a Blender , versión 2.4 y en este la versión 2.5. **Bueno vamos con el tutorial**. -Creamos un nuevo documento , en File/New. -Eliminamos el cubo pulsando la tecla Supr. -Añadimos un Metaball , en Add/Metaball/Metaball , este será el primero de varios. -Añadimos otro Metaball , este no se vé , desplazamos el objeto seleccionado pulsando las flechas de movimiento , hacia arriba , y observamos que los 2 están unidos por una misteriosa fuerza , que si lo alejamos demasiado se separan. Éste es el motivo de utilizar Metaball , en vez de esferas normales.



Uso de Metaball

-Hacemos más grande (escalar) el Metaball seleccionado , dando forma a nuestro objeto. -Así añadiremos los Metaball necesarios para tener la figura que deseamos , en este caso una cabeza , uno para cada oreja , los ojos , la boca.

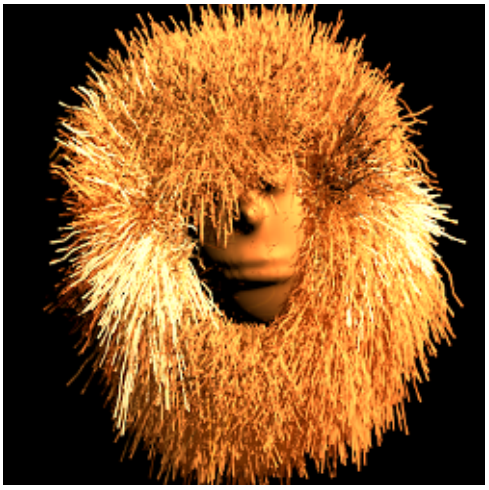


Escalando

-Una vez tengamos una figura (básica) que nos convenza , transformaremos el metaball en un Mesh (Masa), para convertirlo en un objeto con propiedades . Pulsamos Alt + c , y seleccionamos la Opción Mesh from Curve/Meta/… -Ya tenemos un objeto Mesh , con todas sus propiedades , ya podemos asignarle un Material , esculpir la figura , editar los vértices , etc . Ahora vamos a crear 2 materiales , uno será el de la cara y otro será el del pelo. El Material puede ser transparente , tiene 2 colores difuse (color) y especular (color al reflejar), puede reflejar la luz (Mirror) ,etc. -Para crear Pelo vamos a la opción de Partículas , añadimos una , de tipo Hair. Esto nos genera pelo en todas las partes del objeto , para evitar esto , y que solo salga por la parte que nos interesa , osea arriba y detras de la cara, tendremos que crear un grupo de vértices .Vamos a edit mode (o pulsamos Tab) , pulsamos la tecla a, para seleccionar o deseleccionar todos , y pulsando la tecla b nos permite seleccionar creando un rectángulo. Una vez seleccionado pulsamos Ctrl + g para crear un nuevo grupo. En la pestaña de partículas vamos a la opción vertexgroups y en el primero , seleccionamos el grupo creado antes (si es el primero se llama group , sino group0001 , etc).

Uso de Particle Mode

-Ya tenemos el pelo en la zona que queremos.Ahora le asignamos el material 2 ,que es el segundo de la lista de materiales , para que no sea del mismo color que la cara .



Uso de las propiedades de partículas

-Una vez tenemos el pelo del color que tengamos puede que nos interese peinar el pelo, si peinar no me he equivocado , entramo en el modo Particle Mode , a la izquierda en el menú de Brush (brocha) , por defecto arriba está seleccionado none , seleccionamos comb para poder modificar y vamos peinando nuestra figura. -Más o menos tenemos nuestra figura creada , ahora falta renderizar , ¿eso que es? , pués es obtener imágenes de cada paso (frame) , a partir de lo que ve la cámara. En nuestro caso tenemos solo 1 imagen , ya explicaré

como crear animaciones ,sencillas. Si pulsamos Ctrl + Izq de los cursores de teclado o Ctrl + Dcha , podemos cambiar la perspectiva de blender , hay una perspectiva de animación , otra de juegos , y normalmente hay una de cámara. También podemos ver lo que ve la cámara (recordad que es un objeto y la podemos mover , rotar e incluso la podemos animar) pulsando Numpad0 , o en el menú view/Camera. Si seleccionamos la perspectiva de camera podemos ajustar el objeto que hemos creado al rectángulo que ve la cámara , moviendo con las flechas de desplazamiento . Podemos ver el resultado en el menú Render/Render Image , o pulsando F12.

Pelo coloreado y peinado

-Por defecto tenemos una luz , las luces se pueden copiar , modificar el color de la luz , mover , rotar. Como cualquier otro objeto , podemos animar la luz , etc . Para este ejemplo he utilizado la luz de tipo Sun (Sol) , pero podeis elegir la que querais , en el panel Light , estando la luz seleccionada. Por último , un enlace a como crear pelo artístico con **curvas** y la herramienta de **esculpir** : http://simpatiaporblender.blogspot.com/2010/01/interesante-forma-para-modela-cabellos.html No está de más explicar que se puede combinar el pelo creado con el sistema de partículas con pelo creado con el sistema de curvas ,como explica el tutorial anterior , puede dar resultados muy buenos , por ejemplo para

crear una pulsera. y hasta aquí nuestro primer **tutorial** de **blender** .