1 LIBGDX ModelBuilder

UNIDADE 5: ModelBuilder. Primeiros modelos 3D

1.1 Sumario

1.1.1 Introdución

A clase ModelBuilder vainos permitir crear modelos en execución (por programación).

Como novidade con respecto ó dado na Unidade 4 imos engadir o uso de luces.

Preparación: Empezamos co primeiro exercicio.

Preparación: Creade unha nova clase que implemente a interface Screen.

Código da clase EX_1_DefinicionLuz
Obxectivo: Cargar o noso primeiro modelo.

```
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.PerspectiveCamera;


public class EX_1_DefinicionLuz implements Screen {

private PerspectiveCamera camara3d;

public EX_1_DefinicionLuzInicial(){
camara3d = new PerspectiveCamera();

}

@Override
public void render(float delta) {
// TODO Auto-generated method stub
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT | GL20.GL_DEPTH_BUFFER_BIT);

}

@Override
public void resize(int width, int height) {
// TODO Auto-generated method stub

camara3d.fieldOfView=67;
camara3d.viewportWidth=width;
camara3d.viewportHeight=height;

camara3d.position.set(0f,0f,15f);
camara3d.lookAt(0,0,0);
camara3d.near=1;
camara3d.far=300f;
camara3d.update();
```

```
}

@Override
public void show() {
// TODO Auto-generated method stub
}

@Override
public void hide() {
// TODO Auto-generated method stub

}

@Override
public void pause() {
// TODO Auto-generated method stub

}

@Override
public void resume() {
// TODO Auto-generated method stub

}

@Override
public void dispose() {
// TODO Auto-generated method stub

}

}
```

Como vemos nesta clase só definimos unha cámara en perspectiva como xa fixemos anteriormente.

Código da clase PracticasNovaApi3D
Obxectivo: Chamamos á clase anterior

```
import com.badlogic.gdx.Game;
import com.plategaxogo3davanzado.pantallas.EX_1_DefinicionLuzInicial;

public class PracticasNovaApi3D extends Game {

private EX_1_DefinicionLuz pantallaxogo;

@Override
public void create() {
// TODO Auto-generated method stub

pantallaxogo = new EX_1_DefinicionLuz();
setScreen(pantallaxogo);
}

@Override
public void dispose(){
super.dispose();
pantallaxogo.dispose();
}
}
```

Modificade ás clases principais das diferentes plataformas para que chamen a esta clase.

A partires de aquí todos os exercicios levarán consigo a creación dunha clase que implemente a interface Screen e será necesario modificar a clase PracticasNovaApi3D para que chame ó novo exercicio.

1.1.2 ModelBuilder

O obxectivo deste punto vai consistir en xerar unha figura xeométrica por programación e asinarlle unha cor, textura e unha luz á escena

Clase ModelBuilder. Información relacionada na wiki: https://github.com/libgdx/libgdx/wiki/Material-and-environment

O proceso é o seguinte:

- Instanciamos un obxecto da clase ModelBuilder:

```
ModelBuilder modelBuilder = new ModelBuilder();
```

- Chamamos ó método createXXXXX indicando o tipo de figura que queremos crear.

Pero....non todo vai ser tan fácil :)

Cando creamos unha figura temos que asinarlle un material (ven así no constructor). E para que serve un material ?

Os materiais están relacionados coa luz e como se reflexa nela. Dependendo do tipo de material a luz pode reflectirse dunha forma ou outra.

Tedes neste enlace máis información sobre o uso de Materiais e Luces.

Por exemplo, neste caso imos crear un cilindro:

```
private Model modelCilindro;

        ............
        modelCilindro = modelBuilder.createCylinder(5f,5f, 10f, 10,
        new Material(ColorAttribute.createDiffuse(Color.RED)),
        Usage.Position | Usage.Normal);
```

Como xa vimos anteriormente, cando creamos unha figura sempre temos que enviarlle a lo menos á posición. Iso o indicamos có atributo Usage.position. O outro atributo, Ussage.Normal fai referencia a que calcule os vectores que son asociados á figura e que van indicar cara onde se vai a reflectir a luz (normal vector).

Nota: Podedes crear un cubo ou outras figuras para facer probas.

Tedes máis información neste enlace sobre os Vector Normal.

Neste caso estamos a definir un cilindro de dimensións alto=5,ancho=5,profundo=5 e 10 divisións, que veñen a ser o número de partes en que está dividido o cilindro (a mais partes máis 'calidade' visual do cilindro). Ademais o estamos pintando de vermello.

Nota: Débese liberar a memoria do modelo chamando ó método dispose.

Agora ben, có Model non indicamos onde ten que colocarse o figura. Para iso temos que crear unha instancia da clase ModelInstance.

- Creamos por tanto unha instancia da clase ModelInstance, pasando como parámetro o obxecto da clase Model.

```
private ModelInstance modeloInstanciaCilindro;

        ............
public EX_1_DefinicionLuzInicial(){
                modeloInstanciaCilindro = new ModelInstance(modelCilindro);
                .....
        }
```

Nota:Por defecto os obxectos son posicionados na posición (0,0,0).

- Agora necesitamos renderizar o modelo. Para facelo debemos facer uso da clase ModelBatch, chamando ó método render e pasándolle como argumento o ModelInstance:

```
private ModelBatch modelBatch;

        .....
public EX_1_DefinicionLuzInicial(){
                modelBatch = new ModelBatch();
                .....
        }
        .....
@Override
public void render(float delta) {
// TODO Auto-generated method stub
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT | GL20.GL_DEPTH_BUFFER_BIT);

modelBatch.begin(camara3d);

modelBatch.render(modeloInstanciaCilindro);

modelBatch.end();
}
```

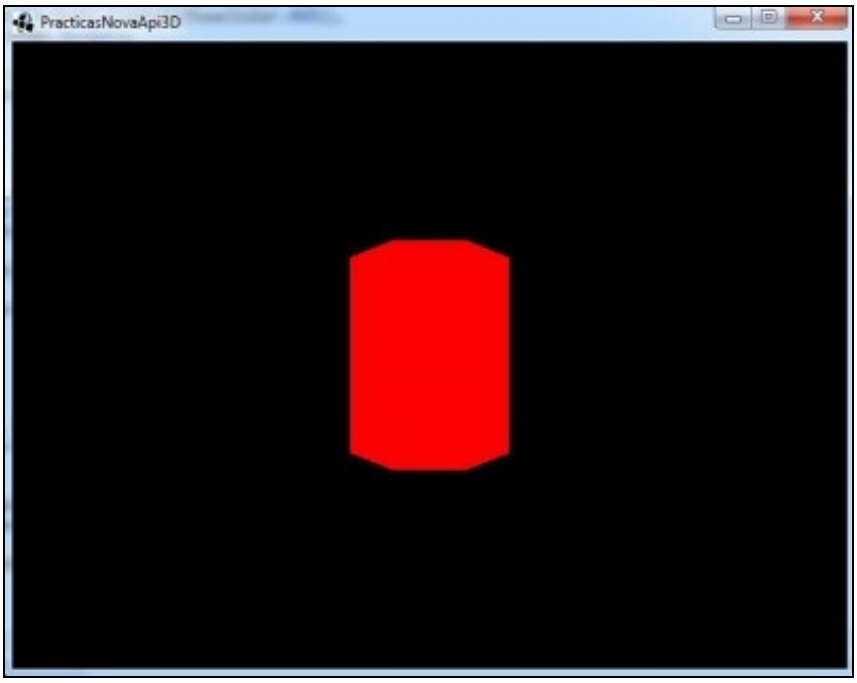
O código completo:

Código da clase EX_1_DefinicionLuzInicial
Obxectivo: Visualiza un cilindro utilizando a clase ModelBuilder.

```
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.PerspectiveCamera;
import com.badlogic.gdx.graphics.VertexAttributes.Usage;
import com.badlogic.gdx.graphics.g3d.Material;
import com.badlogic.gdx.graphics.g3d.Model;
import com.badlogic.gdx.graphics.g3d.ModelBatch;
import com.badlogic.gdx.graphics.g3d.ModelInstance;
import com.badlogic.gdx.graphics.g3d.attributes.ColorAttribute;
import com.badlogic.gdx.graphics.g3d.utils.ModelBuilder;


public class EX_1_DefinicionLuzInicial implements Screen {

private PerspectiveCamera camara3d;
private Model modelCilindro;
private ModelInstance modeloInstanciaCilindro;
private ModelBatch modelBatch;


public EX_1_DefinicionLuzInicial(){
camara3d = new PerspectiveCamera();


        ModelBuilder modelBuilder = new ModelBuilder();


        modelCilindro = modelBuilder.createCylinder(5f,5f, 10f, 10,
        new Material(ColorAttribute.createDiffuse(Color.RED)),
        Usage.Position | Usage.Normal);

        modeloInstanciaCilindro = new ModelInstance(modelCilindro);
        modelBatch = new ModelBatch();

}

@Override
public void render(float delta) {
// TODO Auto-generated method stub
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT | GL20.GL_DEPTH_BUFFER_BIT);

modelBatch.begin(camara3d);
```

```
modelBatch.render(modeloInstanciaCilindro);

modelBatch.end();
}

@Override
public void resize(int width, int height) {
// TODO Auto-generated method stub

camara3d.fieldOfView=67;
camara3d.viewportWidth=width;
camara3d.viewportHeight=height;

camara3d.position.set(0f,0f,15f);
camara3d.lookAt(0,0,0);
camara3d.near=1;
camara3d.far=300f;
camara3d.update();



}

@Override
public void show() {
// TODO Auto-generated method stub
}

@Override
public void hide() {
// TODO Auto-generated method stub

}

@Override
public void pause() {
// TODO Auto-generated method stub

}

@Override
public void resume() {
// TODO Auto-generated method stub

}

@Override
public void dispose() {
// TODO Auto-generated method stub
modelCilindro.dispose();
modelBatch.dispose();


}

}
```

Resultado:



Non é moi espectacular o resultado verdade ? :)

Para facelo un pouco máis bonito (con texturas mellora a cousa) imos darlle un pouco de luz.

1.1.3 Luces

- Máis información: http://sjbaker.org/steve/omniv/opengl_lighting.html

Existen diferentes tipos de fontes de luz:

- Luces de ambiente (ambient lights): Non é un tipo de luz direccional (que veña dun sitio concreto) pero fai que todos os obxectos se iluminen por igual.
- Puntos de luz (point lights): A luz parte dun punto no espazo e todas as direccións (coma unha bombilla).
- Luces direccionais (directional lights): A luz ven dunha dirección. Por exemplo o Sol (aínda que non sexa realmente así). A súa luz 'golpea' toda a superficie da Terra no mesmo ángulo pola distancia tan grande que hai.
- Spotlights: Parecidas ós puntos de luz pero cunha dirección formando un cono de luz, como por exemplo a luz dun farol da rúa.

A maiores da posición e dirección da luz, OPEN GL nos permite definir a intensidade da mesma. Esta é expresada en forma de cor RGBA, e especifica catro tipos diferentes de cor (intensidade) da mesma:

- Ambiente (ambient): A luz é uniforme en todo o obxecto.
- Difusa (diffuse): As caras do obxectos que non son iluminadas aparecen máis escuras.
- Especular (specular): Parecida á difusa pero só afecta a certos puntos que teñen unha determinada orientación entre a luz e o obxecto.
- Emisiva (emissive): Non se utiliza.

Como comentario, indicar que os materias poden ter a mesma cor/intensidade que os tipos indicados anteriormente.

Unha vez vista a teoría imos coa práctica.

Imos engadir á nosa escena unha luz ambiente e unha luz direccional.

Nota: As luces consumen bastantes recursos polo que non se debe abusar delas.

Para facer uso das luces debemos usar a clase Environment.

O proceso é o seguinte:

- Instanciamos un obxecto de dita clase:

```
private Environment environment;
         .............
public EX_1_DefinicionLuzInicial(){
         .............
        environment = new Environment();

        }
```

- Definimos os dous tipos de luz.

```
environment.set(new ColorAttribute(ColorAttribute.AmbientLight, 0.4f, 0.4f, 0.4f, 1f));
        environment.add(new DirectionalLight().set(1f, 1f, 1f, 1f, 0f, 0f));
```

- Renderizamos enviando o environment creado:

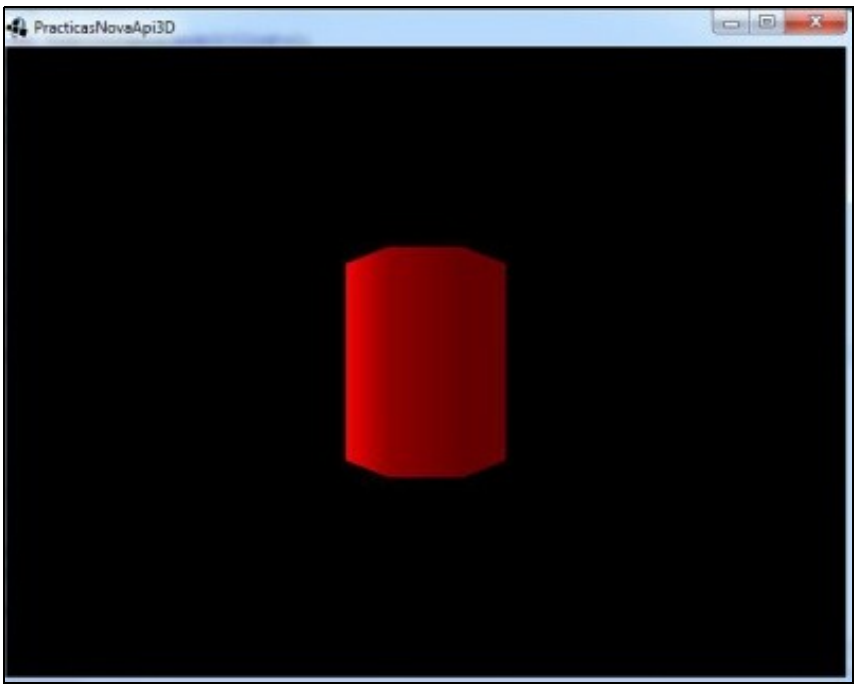
```
public void render(float delta) {
// TODO Auto-generated method stub
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT | GL20.GL_DEPTH_BUFFER_BIT);

modelBatch.begin(camara3d);

modelBatch.render(modeloInstanciaCilindro,environment);

modelBatch.end();
}
```

Como vemos na definición do tipo de luz, a luz ambiente xa está creada. Cambiamos a súa intensidade e non leva posición (ven de todos lados). A luz direccional a engadimos:

```
environment.add(new DirectionalLight().set(1f, 1f, 1f, 1f, 0f, 0f));
```

Indicamos a súa cor/intensidade(RGBA) e posición.

Resultado:



1.1.4 Controlando a escena

Clase CameraInputController.

Para 'xogar' un pouco coa escena imos a engadir un control da cámara co rato, de tal forma que poderemos rotar a cámara e afastala coa roda do rato.

Código da clase EX_1_DefinicionLuz
Obxectivo: Poder mover a cámara premendo co rato o botón esquerdo e movendo o rato ó mesmo tempo. Coa roda achegamos ou afastamos a cámara.

```
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.PerspectiveCamera;
import com.badlogic.gdx.graphics.VertexAttributes.Usage;
import com.badlogic.gdx.graphics.g3d.Environment;
import com.badlogic.gdx.graphics.g3d.Material;
import com.badlogic.gdx.graphics.g3d.Model;
import com.badlogic.gdx.graphics.g3d.ModelBatch;
import com.badlogic.gdx.graphics.g3d.ModelInstance;
import com.badlogic.gdx.graphics.g3d.attributes.ColorAttribute;
import com.badlogic.gdx.graphics.g3d.environment.DirectionalLight;
import com.badlogic.gdx.graphics.g3d.utils.CameraInputController;
import com.badlogic.gdx.graphics.g3d.utils.ModelBuilder;


public class EX_1_DefinicionLuz implements Screen {

private PerspectiveCamera camara3d;
private Model modelCilindro;
private ModelInstance modeloInstanciaCilindro;
private ModelBatch modelBatch;
private Environment environment;

private CameraInputController camController;

public EX_1_DefinicionLuzInicial(){
camara3d = new PerspectiveCamera();


        ModelBuilder modelBuilder = new ModelBuilder();


        modelCilindro = modelBuilder.createCylinder(5f,5f, 10f, 10,
        new Material(ColorAttribute.createDiffuse(Color.RED)),
        Usage.Position | Usage.Normal);

        modeloInstanciaCilindro = new ModelInstance(modelCilindro);
        modelBatch = new ModelBatch();

        environment = new Environment();
        environment.set(new ColorAttribute(ColorAttribute.AmbientLight, 0.4f, 0.4f, 0.4f, 1f));
        environment.add(new DirectionalLight().set(1f, 1f, 1f, 1f, 0f, 0f));

camController = new CameraInputController(camara3d);
Gdx.input.setInputProcessor(camController);

}

@Override
public void render(float delta) {
// TODO Auto-generated method stub
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT | GL20.GL_DEPTH_BUFFER_BIT);

modelBatch.begin(camara3d);

modelBatch.render(modeloInstanciaCilindro,environment);
```

```
modelBatch.end();
}

@Override
public void resize(int width, int height) {
// TODO Auto-generated method stub

camara3d.fieldOfView=67;
camara3d.viewportWidth=width;
camara3d.viewportHeight=height;

camara3d.position.set(0f,0f,15f);
camara3d.lookAt(0,0,0);
camara3d.near=1;
camara3d.far=300f;
camara3d.update();



}

@Override
public void show() {
// TODO Auto-generated method stub
}

@Override
public void hide() {
// TODO Auto-generated method stub

}

@Override
public void pause() {
// TODO Auto-generated method stub

}

@Override
public void resume() {
// TODO Auto-generated method stub

}

@Override
public void dispose() {
// TODO Auto-generated method stub
modelCilindro.dispose();
modelBatch.dispose();


}

}
```