**android-3D-model-viewer/app/src/main/java/org/andresoviedo/app/model3D/view/ModelActivity.java**

package org.andresoviedo.app.model3D.view;

import android.annotation.TargetApi;

import android.app.Activity;

import android.content.ActivityNotFoundException;

import android.content.Intent;

import android.net.Uri;

import android.os.Build;

import android.os.Bundle;

import android.os.Handler;

import android.util.Log;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.widget.Toast;

import org.andresoviedo.app.model3D.demo.ExampleSceneLoader;

import org.andresoviedo.app.model3D.demo.SceneLoader;

import org.andresoviedo.dddmodel2.R;

import org.andresoviedo.util.android.ContentUtils;

import java.io.IOException;

/\*\*

\* This activity represents the container for our 3D viewer.

\*

\* @author andresoviedo

\*/

public class ModelActivity extends Activity {

private static final int REQUEST\_CODE\_LOAD\_TEXTURE = 1000;

private static final int FULLSCREEN\_DELAY = 10000;

/\*\*

\* Type of model if file name has no extension (provided though content provider)

\*/

private int paramType;

/\*\*

\* The file to load. Passed as input parameter

\*/

private Uri paramUri;

/\*\*

\* Enter into Android Immersive mode so the renderer is full screen or not

\*/

private boolean immersiveMode = true;

/\*\*

\* Background GL clear color. Default is light gray

\*/

private float[] backgroundColor = new float[]{0f, 0f, 0f, 1.0f};

private ModelSurfaceView gLView;

private SceneLoader scene;

private Handler handler;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// Try to get input parameters

Bundle b = getIntent().getExtras();

if (b != null) {

if (b.getString("uri") != null) {

this.paramUri = Uri.parse(b.getString("uri"));

}

this.paramType = b.getString("type") != null ? Integer.parseInt(b.getString("type")) : -1;

this.immersiveMode = "true".equalsIgnoreCase(b.getString("immersiveMode"));

try {

String[] backgroundColors = b.getString("backgroundColor").split(" ");

backgroundColor[0] = Float.parseFloat(backgroundColors[0]);

backgroundColor[1] = Float.parseFloat(backgroundColors[1]);

backgroundColor[2] = Float.parseFloat(backgroundColors[2]);

backgroundColor[3] = Float.parseFloat(backgroundColors[3]);

} catch (Exception ex) {

// Assuming default background color

}

}

Log.i("Renderer", "Params: uri '" + paramUri + "'");

handler = new Handler(getMainLooper());

// Create our 3D sceneario

if (paramUri == null) {

scene = new ExampleSceneLoader(this);

} else {

scene = new SceneLoader(this);

}

scene.init();

// Create a GLSurfaceView instance and set it

// as the ContentView for this Activity.

try {

gLView = new ModelSurfaceView(this);

setContentView(gLView);

} catch (Exception e) {

Toast.makeText(this, "Error loading OpenGL view:\n" +e.getMessage(), Toast.LENGTH\_LONG).show();

}

// Show the Up button in the action bar.

setupActionBar();

// TODO: Alert user when there is no multitouch support (2 fingers). He won't be able to rotate or zoom

ContentUtils.printTouchCapabilities(getPackageManager());

setupOnSystemVisibilityChangeListener();

}

/\*\*

\* Set up the {@link android.app.ActionBar}, if the API is available.

\*/

@TargetApi(Build.VERSION\_CODES.HONEYCOMB)

private void setupActionBar() {

// if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.HONEYCOMB) {

// getActionBar().setDisplayHomeAsUpEnabled(true);

// }

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

// Inflate the menu; this adds items to the action bar if it is present.

getMenuInflater().inflate(R.menu.model, menu);

return true;

}

@TargetApi(Build.VERSION\_CODES.JELLY\_BEAN)

private void setupOnSystemVisibilityChangeListener() {

if (Build.VERSION.SDK\_INT < Build.VERSION\_CODES.JELLY\_BEAN) {

return;

}

getWindow().getDecorView().setOnSystemUiVisibilityChangeListener(visibility -> {

// Note that system bars will only be "visible" if none of the

// LOW\_PROFILE, HIDE\_NAVIGATION, or FULLSCREEN flags are set.

if ((visibility & View.SYSTEM\_UI\_FLAG\_FULLSCREEN) == 0) {

// The system bars are visible. Make any desired

hideSystemUIDelayed();

}

});

}

@Override

public void onWindowFocusChanged(boolean hasFocus) {

super.onWindowFocusChanged(hasFocus);

if (hasFocus) {

hideSystemUIDelayed();

}

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.model\_toggle\_wireframe:

scene.toggleWireframe();

break;

case R.id.model\_toggle\_boundingbox:

scene.toggleBoundingBox();

break;

case R.id.model\_toggle\_textures:

scene.toggleTextures();

break;

case R.id.model\_toggle\_animation:

scene.toggleAnimation();

break;

case R.id.model\_toggle\_collision:

scene.toggleCollision();

break;

case R.id.model\_toggle\_lights:

scene.toggleLighting();

break;

case R.id.model\_toggle\_stereoscopic:

scene.toggleStereoscopic();

break;

case R.id.model\_toggle\_blending:

scene.toggleBlending();

break;

case R.id.model\_toggle\_immersive:

toggleImmersive();

break;

case R.id.model\_load\_texture:

Intent target = ContentUtils.createGetContentIntent("image/\*");

Intent intent = Intent.createChooser(target, "Select a file");

try {

startActivityForResult(intent, REQUEST\_CODE\_LOAD\_TEXTURE);

} catch (ActivityNotFoundException e) {

// The reason for the existence of aFileChooser

}

break;

}

hideSystemUIDelayed();

return super.onOptionsItemSelected(item);

}

private void toggleImmersive() {

this.immersiveMode = !this.immersiveMode;

if (Build.VERSION.SDK\_INT < Build.VERSION\_CODES.JELLY\_BEAN) {

return;

}

if (this.immersiveMode) {

hideSystemUI();

} else {

showSystemUI();

}

Toast.makeText(this, "Fullscreen " +this.immersiveMode, Toast.LENGTH\_SHORT).show();

}

private void hideSystemUIDelayed() {

if (!this.immersiveMode) {

return;

}

handler.removeCallbacksAndMessages(null);

handler.postDelayed(this::hideSystemUI, FULLSCREEN\_DELAY);

}

private void hideSystemUI() {

if (!this.immersiveMode) {

return;

}

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.KITKAT) {

hideSystemUIKitKat();

} else if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.JELLY\_BEAN) {

hideSystemUIJellyBean();

}

}

// This snippet hides the system bars.

@TargetApi(Build.VERSION\_CODES.KITKAT)

private void hideSystemUIKitKat() {

// Set the IMMERSIVE flag.

// Set the content to appear under the system bars so that the content

// doesn't resize when the system bars hide and show.

final View decorView = getWindow().getDecorView();

decorView.setSystemUiVisibility(View.SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE | View.SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_NAVIGATION

| View.SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN | View.SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION // hide nav bar

| View.SYSTEM\_UI\_FLAG\_FULLSCREEN // hide status bar

| View.SYSTEM\_UI\_FLAG\_IMMERSIVE);

}

@TargetApi(Build.VERSION\_CODES.JELLY\_BEAN)

private void hideSystemUIJellyBean() {

final View decorView = getWindow().getDecorView();

decorView.setSystemUiVisibility(View.SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE | View.SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_NAVIGATION

| View.SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN | View.SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION

| View.SYSTEM\_UI\_FLAG\_FULLSCREEN | View.SYSTEM\_UI\_FLAG\_LOW\_PROFILE);

}

// This snippet shows the system bars. It does this by removing all the flags

// except for the ones that make the content appear under the system bars.

@TargetApi(Build.VERSION\_CODES.JELLY\_BEAN)

private void showSystemUI() {

handler.removeCallbacksAndMessages(null);

final View decorView = getWindow().getDecorView();

decorView.setSystemUiVisibility(View.SYSTEM\_UI\_FLAG\_VISIBLE);

}

public Uri getParamUri() {

return paramUri;

}

public int getParamType() {

return paramType;

}

public float[] getBackgroundColor() {

return backgroundColor;

}

public SceneLoader getScene() {

return scene;

}

public ModelSurfaceView getGLView() {

return gLView;

}

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

if (resultCode != RESULT\_OK) {

return;

}

switch (requestCode) {

case REQUEST\_CODE\_LOAD\_TEXTURE:

// The URI of the selected file

final Uri uri = data.getData();

if (uri != null) {

Log.i("ModelActivity", "Loading texture '" + uri + "'");

try {

ContentUtils.setThreadActivity(this);

scene.loadTexture(null, uri);

} catch (IOException ex) {

Log.e("ModelActivity", "Error loading texture: " + ex.getMessage(), ex);

Toast.makeText(this, "Error loading texture '" + uri + "'. " + ex

.getMessage(), Toast.LENGTH\_LONG).show();

} finally {

ContentUtils.setThreadActivity(null);

}

}

}

}

}

**android-3D-model-viewer/app/src/main/java/org/andresoviedo/app/model3D/view/MenuActivity.java**

package org.andresoviedo.app.model3D.view;

import android.Manifest;

import android.app.ListActivity;

import android.app.ProgressDialog;

import android.content.ActivityNotFoundException;

import android.content.DialogInterface;

import android.content.Intent;

import android.net.Uri;

import android.os.AsyncTask;

import android.os.Bundle;

import android.util.Log;

import android.view.View;

import android.widget.ArrayAdapter;

import android.widget.ListView;

import android.widget.Toast;

import org.andresoviedo.android\_3d\_model\_engine.services.wavefront.WavefrontLoader;

import org.andresoviedo.dddmodel2.R;

import org.andresoviedo.util.android.AndroidUtils;

import org.andresoviedo.util.android.AssetUtils;

import org.andresoviedo.util.android.ContentUtils;

import org.andresoviedo.util.android.FileUtils;

import org.andresoviedo.util.view.TextActivity;

import java.io.File;

import java.util.HashMap;

import java.util.List;

import java.util.Locale;

import java.util.Map;

public class MenuActivity extends ListActivity {

private static final String REPO\_URL = "https://github.com/andresoviedo/android-3D-model-viewer/raw/master/models/index";

private static final int REQUEST\_READ\_EXTERNAL\_STORAGE = 1000;

private static final int REQUEST\_INTERNET\_ACCESS = 1001;

private static final int REQUEST\_CODE\_OPEN\_FILE = 1101;

private static final int REQUEST\_CODE\_OPEN\_MATERIAL = 1102;

private static final int REQUEST\_CODE\_OPEN\_TEXTURE = 1103;

private static final String SUPPORTED\_FILE\_TYPES\_REGEX = "(?i).\*\\.(obj|stl|dae)";

private enum Action {

LOAD\_MODEL, GITHUB, SETTINGS, HELP, ABOUT, EXIT, UNKNOWN, DEMO

}

/\*\*

\* Load file user data

\*/

private Map<String, Object> loadModelParameters = new HashMap<>();

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_menu);

setListAdapter(new ArrayAdapter<>(this, R.layout.activity\_menu\_item,

getResources().getStringArray(R.array.menu\_items)));

}

@Override

public void onListItemClick(ListView l, View v, int position, long id) {

String selectedItem = (String) getListView().getItemAtPosition(position);

// Toast.makeText(getApplicationContext(), "Click ListItem '" + selectedItem + "'", Toast.LENGTH\_LONG).show();

String selectedAction = selectedItem.replace(' ', '\_').toUpperCase(Locale.getDefault());

Action action = Action.UNKNOWN;

try {

action = Action.valueOf(selectedAction);

} catch (IllegalArgumentException ex) {

Toast.makeText(getApplicationContext(), ex.getMessage(), Toast.LENGTH\_LONG).show();

}

try {

switch (action) {

case DEMO:

Intent demoIntent = new Intent(MenuActivity.this.getApplicationContext(), ModelActivity.class);

demoIntent.putExtra("immersiveMode", "true");

demoIntent.putExtra("backgroundColor", "0 0 0 1");

MenuActivity.this.startActivity(demoIntent);

break;

case GITHUB:

AndroidUtils.openUrl(this, "https://github.com/andresoviedo/android-3D-model-viewer");

break;

case LOAD\_MODEL:

loadModel();

break;

case ABOUT:

Intent aboutIntent = new Intent(MenuActivity.this.getApplicationContext(), TextActivity.class);

aboutIntent.putExtra("title", selectedItem);

aboutIntent.putExtra("text", getResources().getString(R.string.about\_text));

MenuActivity.this.startActivity(aboutIntent);

break;

case HELP:

Intent helpIntent = new Intent(MenuActivity.this.getApplicationContext(), TextActivity.class);

helpIntent.putExtra("title", selectedItem);

helpIntent.putExtra("text", getResources().getString(R.string.help\_text));

MenuActivity.this.startActivity(helpIntent);

break;

case SETTINGS:

break;

case EXIT:

MenuActivity.this.finish();

break;

case UNKNOWN:

Toast.makeText(getApplicationContext(), "Unrecognized action '" + selectedAction + "'",

Toast.LENGTH\_LONG).show();

break;

}

} catch (Exception ex) {

Toast.makeText(getApplicationContext(), ex.getMessage(), Toast.LENGTH\_LONG).show();

}

}

private void loadModel() {

ContentUtils.showListDialog(this, "File Provider", new String[]{"Embedded Models", "App Repository",

"External Storage ", "Content Provider"}, (DialogInterface dialog, int which) -> {

if (which == 0) {

loadModelFromAssets();

} else if (which == 1) {

loadModelFromRepository();

} else if (which == 2) {

loadModelFromSdCard();

} else {

loadModelFromContentProvider();

}

});

}

private void loadModelFromAssets() {

AssetUtils.createChooserDialog(this, "Select file", null, "models", "(?i).\*\\.(obj|stl|dae)",

(String file) -> {

if (file != null) {

ContentUtils.provideAssets(this);

launchModelRendererActivity(Uri.parse("assets://" + getPackageName() + "/" + file));

}

});

}

private void loadModelFromRepository() {

if (!AndroidUtils.checkPermission(this, Manifest.permission.INTERNET, REQUEST\_INTERNET\_ACCESS)) {

return;

}

new LoadRepoIndexTask().execute();

}

class LoadRepoIndexTask extends AsyncTask<Void, Integer, List<String>> {

private final ProgressDialog dialog;

public LoadRepoIndexTask() {

this.dialog = new ProgressDialog(MenuActivity.this);

}

@Override

protected void onPreExecute() {

super.onPreExecute();

this.dialog.setMessage("Loading...");

this.dialog.setCancelable(false);

this.dialog.show();

}

@Override

protected List<String> doInBackground(Void... voids) {

return ContentUtils.getIndex(REPO\_URL);

}

@Override

protected void onPostExecute(List<String> strings) {

if (dialog.isShowing()){

dialog.dismiss();

}

if (strings == null){

Toast.makeText(MenuActivity.this, "Couldn't load repo index", Toast.LENGTH\_LONG).show();

return;

}

ContentUtils.createChooserDialog(MenuActivity.this, "Select file", null,

strings, SUPPORTED\_FILE\_TYPES\_REGEX,

(String file) -> {

if (file != null) {

launchModelRendererActivity(Uri.parse(file));

}

});

}

}

private void loadModelFromSdCard() {

// check permission starting from android API 23 - Marshmallow

if (!AndroidUtils.checkPermission(this, Manifest.permission.READ\_EXTERNAL\_STORAGE, REQUEST\_READ\_EXTERNAL\_STORAGE)) {

return;

}

FileUtils.createChooserDialog(this, "Select file", null, null, SUPPORTED\_FILE\_TYPES\_REGEX,

(File file) -> {

if (file != null) {

ContentUtils.setCurrentDir(file.getParentFile());

launchModelRendererActivity(Uri.parse("file://" + file.getAbsolutePath()));

}

});

}

private void loadModelFromContentProvider() {

loadModelParameters.clear();

ContentUtils.clearDocumentsProvided();

askForFile(REQUEST\_CODE\_OPEN\_FILE, "\*/\*");

}

private void askForFile(int requestCode, String mimeType) {

Intent target = ContentUtils.createGetContentIntent(mimeType);

Intent intent = Intent.createChooser(target, "Select file");

try {

startActivityForResult(intent, requestCode);

} catch (ActivityNotFoundException e) {

Toast.makeText(this, "Error. Please install a file content provider", Toast.LENGTH\_LONG).show();

}

}

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

ContentUtils.setThreadActivity(this);

switch (requestCode) {

case REQUEST\_READ\_EXTERNAL\_STORAGE:

loadModelFromSdCard();

break;

case REQUEST\_INTERNET\_ACCESS:

loadModelFromRepository();

break;

case REQUEST\_CODE\_OPEN\_FILE:

if (resultCode != RESULT\_OK) {

return;

}

final Uri uri = data.getData();

if (uri == null) {

return;

}

// save user selected model

loadModelParameters.put("model", uri);

// detect model type

if (uri.toString().toLowerCase().endsWith(".obj")) {

askForRelatedFiles(0);

} else if (uri.toString().toLowerCase().endsWith(".stl")) {

askForRelatedFiles(1);

} else if (uri.toString().toLowerCase().endsWith(".dae")) {

askForRelatedFiles(2);

} else {

// no model type from filename, ask user...

ContentUtils.showListDialog(this, "Select type", new String[]{"Wavefront (\*.obj)", "Stereolithography (\*" +

".stl)", "Collada (\*.dae)"}, (dialog, which) -> askForRelatedFiles(which));

}

break;

case REQUEST\_CODE\_OPEN\_MATERIAL:

if (resultCode != RESULT\_OK || data.getData() == null) {

launchModelRendererActivity(getUserSelectedModel());

break;

}

String filename = (String) loadModelParameters.get("file");

ContentUtils.addUri(filename, data.getData());

// check if material references texture file

String textureFile = WavefrontLoader.getTextureFile(data.getData());

if (textureFile == null) {

launchModelRendererActivity(getUserSelectedModel());

break;

}

ContentUtils.showDialog(this, "Select texture file", "This model references a " +

"texture file (" + textureFile + "). Please select it", "OK",

"Cancel", (DialogInterface dialog, int which) -> {

switch (which) {

case DialogInterface.BUTTON\_NEGATIVE:

launchModelRendererActivity(getUserSelectedModel());

break;

case DialogInterface.BUTTON\_POSITIVE:

loadModelParameters.put("file", textureFile);

askForFile(REQUEST\_CODE\_OPEN\_TEXTURE, "image/\*");

}

});

break;

case REQUEST\_CODE\_OPEN\_TEXTURE:

if (resultCode != RESULT\_OK || data.getData() == null) {

launchModelRendererActivity(getUserSelectedModel());

break;

}

String textureFilename = (String) loadModelParameters.get("file");

ContentUtils.addUri(textureFilename, data.getData());

launchModelRendererActivity(getUserSelectedModel());

}

}

private Uri getUserSelectedModel() {

return (Uri) loadModelParameters.get("model");

}

private void askForRelatedFiles(int modelType) {

loadModelParameters.put("type", modelType);

switch (modelType) {

case 0: // obj

// check if model references material file

String materialFile = WavefrontLoader.getMaterialLib(getUserSelectedModel());

if (materialFile == null) {

launchModelRendererActivity(getUserSelectedModel());

break;

}

ContentUtils.showDialog(this, "Select material file", "This model references a " +

"material file (" + materialFile + "). Please select it", "OK",

"Cancel", (DialogInterface dialog, int which) -> {

switch (which) {

case DialogInterface.BUTTON\_NEGATIVE:

launchModelRendererActivity(getUserSelectedModel());

break;

case DialogInterface.BUTTON\_POSITIVE:

loadModelParameters.put("file", materialFile);

askForFile(REQUEST\_CODE\_OPEN\_MATERIAL, "\*/\*");

}

});

break;

case 1: // stl

launchModelRendererActivity(getUserSelectedModel());

break;

case 2: // dae

// TODO: pre-process file to ask for referenced textures

// XmlParser.parse(colladaFile)

launchModelRendererActivity(getUserSelectedModel());

break;

}

}

private void launchModelRendererActivity(Uri uri) {

Log.i("Menu", "Launching renderer for '" + uri + "'");

Intent intent = new Intent(getApplicationContext(), ModelActivity.class);

intent.putExtra("uri", uri.toString());

intent.putExtra("immersiveMode", "true");

// content provider case

if (!loadModelParameters.isEmpty()) {

intent.putExtra("type", loadModelParameters.get("type").toString());

loadModelParameters.clear();

}

startActivity(intent);

}

}

**android-3D-model-viewer/app/src/main/java/org/andresoviedo/app/model3D/view/ModelRenderer.java /**

package org.andresoviedo.app.model3D.view;

import android.opengl.GLES20;

import android.opengl.GLSurfaceView;

import android.opengl.Matrix;

import android.util.Log;

import org.andresoviedo.android\_3d\_model\_engine.animation.Animator;

import org.andresoviedo.android\_3d\_model\_engine.drawer.DrawerFactory;

import org.andresoviedo.android\_3d\_model\_engine.model.AnimatedModel;

import org.andresoviedo.android\_3d\_model\_engine.model.Camera;

import org.andresoviedo.android\_3d\_model\_engine.model.Object3D;

import org.andresoviedo.android\_3d\_model\_engine.model.Object3DData;

import org.andresoviedo.android\_3d\_model\_engine.services.Object3DBuilder;

import org.andresoviedo.app.model3D.demo.SceneLoader;

import org.andresoviedo.util.android.GLUtil;

import java.io.ByteArrayInputStream;

import java.io.IOException;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import javax.microedition.khronos.egl.EGLConfig;

import javax.microedition.khronos.opengles.GL10;

public class ModelRenderer implements GLSurfaceView.Renderer {

private final static String TAG = ModelRenderer.class.getName();

/\*\*

\* Add 0.5f to the alpha component to the global shader so we can see through the skin

\*/

private static final float[] BLENDING\_FORCED\_MASK\_COLOR = {1.0f, 1.0f, 1.0f, 0.5f};

// frustrum - nearest pixel

private static final float near = 1f;

// frustrum - fartest pixel

private static final float far = 100f;

// stereoscopic variables

private static float EYE\_DISTANCE = 0.64f;

private static final float[] COLOR\_RED = {1.0f, 0.0f, 0.0f, 1f};

private static final float[] COLOR\_BLUE = {0.0f, 1.0f, 0.0f, 1f};

// 3D window (parent component)

private ModelSurfaceView main;

// width of the screen

private int width;

// height of the screen

private int height;

/\*\*

\* Drawer factory to get right renderer/shader based on object attributes

\*/

private DrawerFactory drawer;

/\*\*

\* 3D Axis (to show if needed)

\*/

private final Object3DData axis = Object3DBuilder.buildAxis().setId("axis");

// The wireframe associated shape (it should be made of lines only)

private Map<Object3DData, Object3DData> wireframes = new HashMap<>();

// The loaded textures

private Map<Object, Integer> textures = new HashMap<>();

// The corresponding opengl bounding boxes and drawer

private Map<Object3DData, Object3DData> boundingBoxes = new HashMap<>();

// The corresponding opengl bounding boxes

private Map<Object3DData, Object3DData> normals = new HashMap<>();

private Map<Object3DData, Object3DData> skeleton = new HashMap<>();

// 3D matrices to project our 3D world

private final float[] viewMatrix = new float[16];

private final float[] modelViewMatrix = new float[16];

private final float[] projectionMatrix = new float[16];

private final float[] viewProjectionMatrix = new float[16];

private final float[] lightPosInWorldSpace = new float[4];

private final float[] cameraPosInWorldSpace = new float[3];

// 3D stereoscopic matrix (left & right camera)

private final float[] viewMatrixLeft = new float[16];

private final float[] projectionMatrixLeft = new float[16];

private final float[] viewProjectionMatrixLeft = new float[16];

private final float[] viewMatrixRight = new float[16];

private final float[] projectionMatrixRight = new float[16];

private final float[] viewProjectionMatrixRight = new float[16];

/\*\*

\* Whether the info of the model has been written to console log

\*/

private Map<Object3DData, Boolean> infoLogged = new HashMap<>();

/\*\*

\* Switch to akternate drawing of right and left image

\*/

private boolean anaglyphSwitch = false;

/\*\*

\* Skeleton Animator

\*/

private Animator animator = new Animator();

/\*\*

\* Did the application explode?

\*/

private boolean fatalException = false;

/\*\*

\* Construct a new renderer for the specified surface view

\*

\* @param modelSurfaceView

\* the 3D window

\*/

public ModelRenderer(ModelSurfaceView modelSurfaceView) throws IllegalAccessException, IOException {

this.main = modelSurfaceView;

// This component will draw the actual models using OpenGL

drawer = new DrawerFactory(modelSurfaceView.getContext());

}

public float getNear() {

return near;

}

public float getFar() {

return far;

}

@Override

public void onSurfaceCreated(GL10 unused, EGLConfig config) {

// Set the background frame color

float[] backgroundColor = main.getModelActivity().getBackgroundColor();

GLES20.glClearColor(backgroundColor[0], backgroundColor[1], backgroundColor[2], backgroundColor[3]);

// Use culling to remove back faces.

// Don't remove back faces so we can see them

// GLES20.glEnable(GLES20.GL\_CULL\_FACE);

// Enable depth testing for hidden-surface elimination.

GLES20.glEnable(GLES20.GL\_DEPTH\_TEST);

// Enable not drawing out of view port

GLES20.glEnable(GLES20.GL\_SCISSOR\_TEST);

}

@Override

public void onSurfaceChanged(GL10 unused, int width, int height) {

this.width = width;

this.height = height;

// Adjust the viewport based on geometry changes, such as screen rotation

GLES20.glViewport(0, 0, width, height);

// the projection matrix is the 3D virtual space (cube) that we want to project

float ratio = (float) width / height;

Log.d(TAG, "projection: [" + -ratio + "," + ratio + ",-1,1]-near/far[1,10]");

Matrix.frustumM(projectionMatrix, 0, -ratio, ratio, -1, 1, getNear(), getFar());

Matrix.frustumM(projectionMatrixRight, 0, -ratio, ratio, -1, 1, getNear(), getFar());

Matrix.frustumM(projectionMatrixLeft, 0, -ratio, ratio, -1, 1, getNear(), getFar());

}

@Override

public void onDrawFrame(GL10 unused) {

if(fatalException){

return;

}

try {

GLES20.glViewport(0, 0, width, height);

GLES20.glScissor(0, 0, width, height);

// Draw background color

GLES20.glClear(GLES20.GL\_COLOR\_BUFFER\_BIT | GLES20.GL\_DEPTH\_BUFFER\_BIT);

SceneLoader scene = main.getModelActivity().getScene();

if (scene == null) {

// scene not ready

return;

}

float[] colorMask = null;

if (scene.isBlendingEnabled()) {

// Enable blending for combining colors when there is transparency

GLES20.glEnable(GLES20.GL\_BLEND);

GLES20.glBlendFunc(GLES20.GL\_ONE, GLES20.GL\_ONE\_MINUS\_SRC\_ALPHA);

if (scene.isBlendingForced()){

colorMask = BLENDING\_FORCED\_MASK\_COLOR;

}

} else {

GLES20.glDisable(GLES20.GL\_BLEND);

}

// animate scene

scene.onDrawFrame();

// recalculate mvp matrix according to where we are looking at now

Camera camera = scene.getCamera();

cameraPosInWorldSpace[0]=camera.xPos;

cameraPosInWorldSpace[1]=camera.yPos;

cameraPosInWorldSpace[2]=camera.zPos;

if (camera.hasChanged()) {

// INFO: Set the camera position (View matrix)

// The camera has 3 vectors (the position, the vector where we are looking at, and the up position (sky)

// the projection matrix is the 3D virtual space (cube) that we want to project

float ratio = (float) width / height;

// Log.v(TAG, "Camera changed: projection: [" + -ratio + "," + ratio + ",-1,1]-near/far[1,10], ");

if (!scene.isStereoscopic()) {

Matrix.setLookAtM(viewMatrix, 0, camera.xPos, camera.yPos, camera.zPos, camera.xView, camera.yView,

camera.zView, camera.xUp, camera.yUp, camera.zUp);

Matrix.multiplyMM(viewProjectionMatrix, 0, projectionMatrix, 0, viewMatrix, 0);

} else {

Camera[] stereoCamera = camera.toStereo(EYE\_DISTANCE);

Camera leftCamera = stereoCamera[0];

Camera rightCamera = stereoCamera[1];

// camera on the left for the left eye

Matrix.setLookAtM(viewMatrixLeft, 0, leftCamera.xPos, leftCamera.yPos, leftCamera.zPos, leftCamera

.xView,

leftCamera.yView, leftCamera.zView, leftCamera.xUp, leftCamera.yUp, leftCamera.zUp);

// camera on the right for the right eye

Matrix.setLookAtM(viewMatrixRight, 0, rightCamera.xPos, rightCamera.yPos, rightCamera.zPos, rightCamera

.xView,

rightCamera.yView, rightCamera.zView, rightCamera.xUp, rightCamera.yUp, rightCamera.zUp);

if (scene.isAnaglyph()) {

Matrix.frustumM(projectionMatrixRight, 0, -ratio, ratio, -1, 1, getNear(), getFar());

Matrix.frustumM(projectionMatrixLeft, 0, -ratio, ratio, -1, 1, getNear(), getFar());

} else if (scene.isVRGlasses()) {

float ratio2 = (float) width / 2 / height;

Matrix.frustumM(projectionMatrixRight, 0, -ratio2, ratio2, -1, 1, getNear(), getFar());

Matrix.frustumM(projectionMatrixLeft, 0, -ratio2, ratio2, -1, 1, getNear(), getFar());

}

// Calculate the projection and view transformation

Matrix.multiplyMM(viewProjectionMatrixLeft, 0, projectionMatrixLeft, 0, viewMatrixLeft, 0);

Matrix.multiplyMM(viewProjectionMatrixRight, 0, projectionMatrixRight, 0, viewMatrixRight, 0);

}

camera.setChanged(false);

}

if (!scene.isStereoscopic()) {

this.onDrawFrame(viewMatrix, projectionMatrix, viewProjectionMatrix, lightPosInWorldSpace, colorMask, cameraPosInWorldSpace);

return;

}

if (scene.isAnaglyph()) {

// INFO: switch because blending algorithm doesn't mix colors

if (anaglyphSwitch) {

this.onDrawFrame(viewMatrixLeft, projectionMatrixLeft, viewProjectionMatrixLeft, lightPosInWorldSpace,

COLOR\_RED, cameraPosInWorldSpace);

} else {

this.onDrawFrame(viewMatrixRight, projectionMatrixRight, viewProjectionMatrixRight, lightPosInWorldSpace,

COLOR\_BLUE, cameraPosInWorldSpace);

}

anaglyphSwitch = !anaglyphSwitch;

return;

}

if (scene.isVRGlasses()) {

// draw left eye image

GLES20.glViewport(0, 0, width / 2, height);

GLES20.glScissor(0, 0, width / 2, height);

this.onDrawFrame(viewMatrixLeft, projectionMatrixLeft, viewProjectionMatrixLeft, lightPosInWorldSpace,

null, cameraPosInWorldSpace);

// draw right eye image

GLES20.glViewport(width / 2, 0, width / 2, height);

GLES20.glScissor(width / 2, 0, width / 2, height);

this.onDrawFrame(viewMatrixRight, projectionMatrixRight, viewProjectionMatrixRight, lightPosInWorldSpace,

null, cameraPosInWorldSpace);

}

}catch (Exception ex){

Log.e("ModelRenderer", "Fatal exception: "+ex.getMessage(), ex);

fatalException = true;

}

}

private void onDrawFrame(float[] viewMatrix, float[] projectionMatrix, float[] viewProjectionMatrix,

float[] lightPosInWorldSpace, float[] colorMask, float[] cameraPosInWorldSpace) {

SceneLoader scene = main.getModelActivity().getScene();

// draw light

if (scene.isDrawLighting()) {

Object3D lightBulbDrawer = drawer.getPointDrawer();

// Calculate position of the light in world space to support lighting

if (scene.isRotatingLight()) {

Matrix.multiplyMV(lightPosInWorldSpace, 0, scene.getLightBulb().getModelMatrix(), 0, scene.getLightPosition(), 0);

// Draw a point that represents the light bulb

lightBulbDrawer.draw(scene.getLightBulb(), projectionMatrix, viewMatrix, -1, lightPosInWorldSpace,

colorMask, cameraPosInWorldSpace);

} else {

lightPosInWorldSpace[0] = scene.getCamera().xPos;

lightPosInWorldSpace[1] = scene.getCamera().yPos;

lightPosInWorldSpace[2] = scene.getCamera().zPos;

lightPosInWorldSpace[3] = 0;

}

// FIXME: memory leak

if (scene.isDrawNormals()) {

lightBulbDrawer.draw(Object3DBuilder.buildLine(new float[]{lightPosInWorldSpace[0],

lightPosInWorldSpace[1], lightPosInWorldSpace[2], 0, 0, 0}), projectionMatrix,

viewMatrix, -1,

lightPosInWorldSpace,

colorMask, cameraPosInWorldSpace);

}

}

// draw axis

if (scene.isDrawAxis()){

Object3D basicDrawer = drawer.getPointDrawer();

basicDrawer.draw(axis, projectionMatrix, viewMatrix, axis.getDrawMode(), axis

.getDrawSize(),-1, lightPosInWorldSpace, colorMask, cameraPosInWorldSpace);

}

// is there any object?

if (scene.getObjects().isEmpty()){

return;

}

// draw all available objects

List<Object3DData> objects = scene.getObjects();

for (int i=0; i<objects.size(); i++) {

Object3DData objData = null;

try {

objData = objects.get(i);

if (!objData.isVisible()) continue;

Object3D drawerObject = drawer.getDrawer(objData, scene.isDrawTextures(), scene.isDrawLighting(),

scene.isDoAnimation(), scene.isDrawColors());

if (drawerObject == null){

continue;

}

if (!infoLogged.containsKey(objData)) {

Log.v("ModelRenderer","Drawing model: "+objData.getId());

infoLogged.put(objData, true);

}

boolean changed = objData.isChanged();

// load model texture

Integer textureId = textures.get(objData.getTextureData());

if (textureId == null && objData.getTextureData() != null) {

Log.i("ModelRenderer","Loading texture '"+objData.getTextureFile()+"'...");

ByteArrayInputStream textureIs = new ByteArrayInputStream(objData.getTextureData());

textureId = GLUtil.loadTexture(textureIs);

textureIs.close();

textures.put(objData.getTextureData(), textureId);

Log.i("GLUtil", "Loaded texture ok. id: "+textureId);

}

if (textureId == null){

textureId = -1;

}

// draw points

if (objData.getDrawMode() == GLES20.GL\_POINTS){

Object3D basicDrawer = drawer.getPointDrawer();

basicDrawer.draw(objData, projectionMatrix, viewMatrix, GLES20.GL\_POINTS, lightPosInWorldSpace, cameraPosInWorldSpace);

}

// draw wireframe

else if (scene.isDrawWireframe() && objData.getDrawMode() != GLES20.GL\_POINTS

&& objData.getDrawMode() != GLES20.GL\_LINES && objData.getDrawMode() != GLES20.GL\_LINE\_STRIP

&& objData.getDrawMode() != GLES20.GL\_LINE\_LOOP) {

// Log.d("ModelRenderer","Drawing wireframe model...");

try{

// Only draw wireframes for objects having faces (triangles)

Object3DData wireframe = wireframes.get(objData);

if (wireframe == null || changed) {

Log.i("ModelRenderer","Generating wireframe model...");

wireframe = Object3DBuilder.buildWireframe(objData);

wireframes.put(objData, wireframe);

}

drawerObject.draw(wireframe, projectionMatrix, viewMatrix, wireframe.getDrawMode(),

wireframe.getDrawSize(), textureId, lightPosInWorldSpace,

colorMask, cameraPosInWorldSpace);

animator.update(wireframe, scene.isShowBindPose());

}catch(Error e){

Log.e("ModelRenderer",e.getMessage(),e);

}

}

// draw points

else if (scene.isDrawPoints() || objData.getFaces() == null || !objData.getFaces().loaded()){

drawerObject.draw(objData, projectionMatrix, viewMatrix

, GLES20.GL\_POINTS, objData.getDrawSize(),

textureId, lightPosInWorldSpace, colorMask, cameraPosInWorldSpace);

}

// draw skeleton

else if (scene.isDrawSkeleton() && objData instanceof AnimatedModel && ((AnimatedModel) objData)

.getAnimation() != null){

Object3DData skeleton = this.skeleton.get(objData);

if (skeleton == null){

skeleton = Object3DBuilder.buildSkeleton((AnimatedModel) objData);

this.skeleton.put(objData, skeleton);

}

animator.update(skeleton, scene.isShowBindPose());

drawerObject = drawer.getDrawer(skeleton, false, scene.isDrawLighting(), scene

.isDoAnimation(), scene.isDrawColors());

drawerObject.draw(skeleton, projectionMatrix, viewMatrix,-1, lightPosInWorldSpace, colorMask, cameraPosInWorldSpace);

}

// draw solids

else {

drawerObject.draw(objData, projectionMatrix, viewMatrix,

textureId, lightPosInWorldSpace, colorMask, cameraPosInWorldSpace);

}

// Draw bounding box

if (scene.isDrawBoundingBox() || scene.getSelectedObject() == objData) {

Object3DData boundingBoxData = boundingBoxes.get(objData);

if (boundingBoxData == null || changed) {

boundingBoxData = Object3DBuilder.buildBoundingBox(objData);

boundingBoxes.put(objData, boundingBoxData);

}

Object3D boundingBoxDrawer = drawer.getBoundingBoxDrawer();

boundingBoxDrawer.draw(boundingBoxData, projectionMatrix, viewMatrix, -1,

lightPosInWorldSpace, colorMask, cameraPosInWorldSpace);

}

// Draw normals

if (scene.isDrawNormals()) {

Object3DData normalData = normals.get(objData);

if (normalData == null || changed) {

normalData = Object3DBuilder.buildFaceNormals(objData);

if (normalData != null) {

// it can be null if object isnt made of triangles

normals.put(objData, normalData);

}

}

if (normalData != null) {

Object3D normalsDrawer = drawer.getDrawer(normalData,false,false,scene.isDoAnimation(),

false);

animator.update(normalData, scene.isShowBindPose());

normalsDrawer.draw(normalData, projectionMatrix, viewMatrix, -1, null,

lightPosInWorldSpace, cameraPosInWorldSpace);

}

}

// TODO: enable this only when user wants it

// obj3D.drawVectorNormals(result, viewMatrix);

} catch (Exception ex) {

Log.e("ModelRenderer","There was a problem rendering the object '"+objData.getId()+"':"+ex.getMessage(),ex);

}

}

}

public int getWidth() {

return width;

}

public int getHeight() {

return height;

}

public float[] getModelProjectionMatrix() {

return projectionMatrix;

}

public float[] getModelViewMatrix() {

return viewMatrix;

}

}

android-3D-model-viewer/app/src/main/java/org/andresoviedo/app/model3D/view/ModelSurfaceView.java /

package org.andresoviedo.app.model3D.view;

import android.opengl.GLSurfaceView;

import android.view.MotionEvent;

import org.andresoviedo.app.model3D.controller.TouchController;

import java.io.IOException;

/\*\*

\* This is the actual opengl view. From here we can detect touch gestures for example

\*

\* @author andresoviedo

\*

\*/

public class ModelSurfaceView extends GLSurfaceView {

private ModelActivity parent;

private ModelRenderer mRenderer;

private TouchController touchHandler;

public ModelSurfaceView(ModelActivity parent) throws IllegalAccessException, IOException {

super(parent);

// parent component

this.parent = parent;

// Create an OpenGL ES 2.0 context.

setEGLContextClientVersion(2);

// This is the actual renderer of the 3D space

mRenderer = new ModelRenderer(this);

setRenderer(mRenderer);

// Render the view only when there is a change in the drawing data

// TODO: enable this?

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

touchHandler = new TouchController(this, mRenderer);

}

@Override

public boolean onTouchEvent(MotionEvent event) {

return touchHandler.onTouchEvent(event);

}

public ModelActivity getModelActivity() {

return parent;

}

public ModelRenderer getModelRenderer(){

return mRenderer;

}

}

**android-3D-model-viewer/app/src/main/java/org/andresoviedo/app/model3D/MainActivity.java /**

package org.andresoviedo.app.model3D;

import android.app.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.view.Menu;

import org.andresoviedo.app.model3D.view.MenuActivity;

import org.andresoviedo.app.model3D.view.ModelActivity;

import org.andresoviedo.dddmodel2.R;

import org.andresoviedo.util.android.AndroidURLStreamHandlerFactory;

import java.net.URL;

/\*\*

\* This is the main android activity. From here we launch the whole stuff.

\*

\* Basically, this activity may serve to show a Splash screen and copy the assets (obj models) from the jar to external

\* directory.

\*

\* @author andresoviedo

\*

\*/

public class MainActivity extends Activity {

// Custom handler: org/andresoviedo/util/android/assets/Handler.class

static {

System.setProperty("java.protocol.handler.pkgs", "org.andresoviedo.util.android");

URL.setURLStreamHandlerFactory(new AndroidURLStreamHandlerFactory());

}

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// Set main layout controls.

// Basically, this is a screen with the app name just in the middle of the scree

setContentView(R.layout.activity\_main);

// Start Model activity.

MainActivity.this.startActivity(new Intent(MainActivity.this.getApplicationContext(), MenuActivity.class));

MainActivity.this.finish();

}

@SuppressWarnings("unused")

private void init() {

MainActivity.this.startActivity(new Intent(MainActivity.this.getApplicationContext(), ModelActivity.class));

MainActivity.this.finish();

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

// Inflate the menu; this adds items to the action bar if it is present.

getMenuInflater().inflate(R.menu.main, menu);

return true;

}

}

**android-3D-model-viewer/app/src/main/java/org/andresoviedo/app/model3D/demo/SceneLoader.java**

package org.andresoviedo.app.model3D.demo;

import android.net.Uri;

import android.os.SystemClock;

import android.util.Log;

import android.widget.Toast;

import org.andresoviedo.android\_3d\_model\_engine.animation.Animator;

import org.andresoviedo.android\_3d\_model\_engine.collision.CollisionDetection;

import org.andresoviedo.android\_3d\_model\_engine.model.Camera;

import org.andresoviedo.android\_3d\_model\_engine.model.Object3DData;

import org.andresoviedo.android\_3d\_model\_engine.services.LoaderTask;

import org.andresoviedo.android\_3d\_model\_engine.services.Object3DBuilder;

import org.andresoviedo.android\_3d\_model\_engine.services.collada.ColladaLoaderTask;

import org.andresoviedo.android\_3d\_model\_engine.services.stl.STLLoaderTask;

import org.andresoviedo.android\_3d\_model\_engine.services.wavefront.WavefrontLoaderTask;

import org.andresoviedo.app.model3D.view.ModelActivity;

import org.andresoviedo.app.model3D.view.ModelRenderer;

import org.andresoviedo.util.android.ContentUtils;

import org.andresoviedo.util.io.IOUtils;

import java.io.IOException;

import java.io.InputStream;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

/\*\*

\* This class loads a 3D scena as an example of what can be done with the app

\*

\* @author andresoviedo

\*/

public class SceneLoader implements LoaderTask.Callback {

/\*\*

\* Default model color: yellow

\*/

private static float[] DEFAULT\_COLOR = {1.0f, 1.0f, 0, 1.0f};

/\*\*

\* Parent component

\*/

protected final ModelActivity parent;

/\*\*

\* List of data objects containing info for building the opengl objects

\*/

private List<Object3DData> objects = new ArrayList<>();

/\*\*

\* Show axis or not

\*/

private boolean drawAxis = false;

/\*\*

\* Point of view camera

\*/

private Camera camera;

/\*\*

\* Enable or disable blending (transparency)

\*/

private boolean isBlendingEnabled = true;

/\*\*

\* Force transparency

\*/

private boolean isBlendingForced = false;

/\*\*

\* Whether to draw objects as wireframes

\*/

private boolean drawWireframe = false;

/\*\*

\* Whether to draw using points

\*/

private boolean drawingPoints = false;

/\*\*

\* Whether to draw bounding boxes around objects

\*/

private boolean drawBoundingBox = false;

/\*\*

\* Whether to draw face normals. Normally used to debug models

\*/

// TODO: toggle feature this

private boolean drawNormals = false;

/\*\*

\* Whether to draw using textures

\*/

private boolean drawTextures = true;

/\*\*

\* Whether to draw using colors or use default white color

\*/

private boolean drawColors = true;

/\*\*

\* Light toggle feature: we have 3 states: no light, light, light + rotation

\*/

private boolean rotatingLight = true;

/\*\*

\* Light toggle feature: whether to draw using lights

\*/

private boolean drawLighting = true;

/\*\*

\* Animate model (dae only) or not

\*/

private boolean doAnimation = true;

/\*\*

\* show bind pose only

\*/

private boolean showBindPose = false;

/\*\*

\* Draw skeleton or not

\*/

private boolean drawSkeleton = false;

/\*\*

\* Toggle collision detection

\*/

private boolean isCollision = false;

/\*\*

\* Toggle 3d

\*/

private boolean isStereoscopic = false;

/\*\*

\* Toggle 3d anaglyph (red, blue glasses)

\*/

private boolean isAnaglyph = false;

/\*\*

\* Toggle 3d VR glasses

\*/

private boolean isVRGlasses = false;

/\*\*

\* Object selected by the user

\*/

private Object3DData selectedObject = null;

/\*\*

\* Initial light position

\*/

private final float[] lightPosition = new float[]{0, 0, 6, 1};

/\*\*

\* Light bulb 3d data

\*/

private final Object3DData lightPoint = Object3DBuilder.buildPoint(lightPosition).setId("light");

/\*\*

\* Animator

\*/

private Animator animator = new Animator();

/\*\*

\* Did the user touched the model for the first time?

\*/

private boolean userHasInteracted;

/\*\*

\* time when model loading has started (for stats)

\*/

private long startTime;

public SceneLoader(ModelActivity main) {

this.parent = main;

}

public void init() {

// Camera to show a point of view

camera = new Camera();

camera.setChanged(true); // force first draw

if (parent.getParamUri() == null){

return;

}

startTime = SystemClock.uptimeMillis();

Uri uri = parent.getParamUri();

Log.i("Object3DBuilder", "Loading model " + uri + ". async and parallel..");

if (uri.toString().toLowerCase().endsWith(".obj") || parent.getParamType() == 0) {

new WavefrontLoaderTask(parent, uri, this).execute();

} else if (uri.toString().toLowerCase().endsWith(".stl") || parent.getParamType() == 1) {

Log.i("Object3DBuilder", "Loading STL object from: "+uri);

new STLLoaderTask(parent, uri, this).execute();

} else if (uri.toString().toLowerCase().endsWith(".dae") || parent.getParamType() == 2) {

Log.i("Object3DBuilder", "Loading Collada object from: "+uri);

new ColladaLoaderTask(parent, uri, this).execute();

}

}

public boolean isDrawAxis(){

return drawAxis;

}

public void setDrawAxis(boolean drawAxis) {

this.drawAxis = drawAxis;

}

public Camera getCamera() {

return camera;

}

private void makeToastText(final String text, final int toastDuration) {

parent.runOnUiThread(() -> Toast.makeText(parent.getApplicationContext(), text, toastDuration).show());

}

public Object3DData getLightBulb() {

return lightPoint;

}

public float[] getLightPosition() {

return lightPosition;

}

/\*\*

\* Hook for animating the objects before the rendering

\*/

public void onDrawFrame() {

animateLight();

// smooth camera transition

camera.animate();

// initial camera animation. animate if user didn't touch the screen

if (!userHasInteracted) {

animateCamera();

}

if (objects.isEmpty()) return;

if (doAnimation) {

for (int i=0; i<objects.size(); i++) {

Object3DData obj = objects.get(i);

animator.update(obj, isShowBindPose());

}

}

}

private void animateLight() {

if (!rotatingLight) return;

// animate light - Do a complete rotation every 5 seconds.

long time = SystemClock.uptimeMillis() % 5000L;

float angleInDegrees = (360.0f / 5000.0f) \* ((int) time);

lightPoint.setRotationY(angleInDegrees);

}

private void animateCamera(){

camera.translateCamera(0.0025f, 0f);

}

synchronized void addObject(Object3DData obj) {

List<Object3DData> newList = new ArrayList<Object3DData>(objects);

newList.add(obj);

this.objects = newList;

requestRender();

}

private void requestRender() {

// request render only if GL view is already initialized

if (parent.getGLView() != null) {

parent.getGLView().requestRender();

}

}

public synchronized List<Object3DData> getObjects() {

return objects;

}

public void toggleWireframe() {

if (!this.drawWireframe && !this.drawingPoints && !this.drawSkeleton){

this.drawWireframe = true;

makeToastText("Wireframe", Toast.LENGTH\_SHORT);

} else if (!this.drawingPoints && !this.drawSkeleton){

this.drawWireframe = false;

this.drawingPoints = true;

makeToastText("Points", Toast.LENGTH\_SHORT);

} else if (!this.drawSkeleton){

this.drawingPoints = false;

this.drawSkeleton = true;

makeToastText("Skeleton", Toast.LENGTH\_SHORT);

} else {

this.drawSkeleton = false;

makeToastText("Faces", Toast.LENGTH\_SHORT);

}

requestRender();

}

public boolean isDrawWireframe() {

return this.drawWireframe;

}

public boolean isDrawPoints() {

return this.drawingPoints;

}

public void toggleBoundingBox() {

this.drawBoundingBox = !drawBoundingBox;

requestRender();

}

public boolean isDrawBoundingBox() {

return drawBoundingBox;

}

public boolean isDrawNormals() {

return drawNormals;

}

public void toggleTextures() {

if (drawTextures && drawColors){

this.drawTextures = false;

this.drawColors = true;

makeToastText("Texture off", Toast.LENGTH\_SHORT);

} else if (drawColors){

this.drawTextures = false;

this.drawColors = false;

makeToastText("Colors off", Toast.LENGTH\_SHORT);

} else {

this.drawTextures = true;

this.drawColors = true;

makeToastText("Textures on", Toast.LENGTH\_SHORT);

}

}

public void toggleLighting() {

if (this.drawLighting && this.rotatingLight) {

this.rotatingLight = false;

makeToastText("Light stopped", Toast.LENGTH\_SHORT);

} else if (this.drawLighting && !this.rotatingLight) {

this.drawLighting = false;

makeToastText("Lights off", Toast.LENGTH\_SHORT);

} else {

this.drawLighting = true;

this.rotatingLight = true;

makeToastText("Light on", Toast.LENGTH\_SHORT);

}

requestRender();

}

public void toggleAnimation() {

if (!this.doAnimation){

this.doAnimation = true;

this.showBindPose = false;

makeToastText("Animation on", Toast.LENGTH\_SHORT);

} else {

this.doAnimation = false;

this.showBindPose = true;

makeToastText("Bind pose", Toast.LENGTH\_SHORT);

}

}

public boolean isDoAnimation() {

return doAnimation;

}

public boolean isShowBindPose() {

return showBindPose;

}

public void toggleCollision() {

this.isCollision = !isCollision;

makeToastText("Collisions: "+isCollision, Toast.LENGTH\_SHORT);

}

public void toggleStereoscopic() {

if (!this.isStereoscopic){

this.isStereoscopic = true;

this.isAnaglyph = true;

this.isVRGlasses = false;

makeToastText("Stereoscopic Anaplygh", Toast.LENGTH\_SHORT);

} else if (this.isAnaglyph){

this.isAnaglyph = false;

this.isVRGlasses = true;

// move object automatically cause with VR glasses we still have no way of moving object

this.userHasInteracted = false;

makeToastText("Stereoscopic VR Glasses", Toast.LENGTH\_SHORT);

} else {

this.isStereoscopic = false;

this.isAnaglyph = false;

this.isVRGlasses = false;

makeToastText("Stereoscopic disabled", Toast.LENGTH\_SHORT);

}

// recalculate camera

this.camera.setChanged(true);

}

public boolean isVRGlasses() {

return isVRGlasses;

}

public boolean isDrawTextures() {

return drawTextures;

}

public boolean isDrawColors() {

return drawColors;

}

public boolean isDrawLighting() {

return drawLighting;

}

public boolean isDrawSkeleton() {

return drawSkeleton;

}

public boolean isCollision() {

return isCollision;

}

public boolean isStereoscopic() {

return isStereoscopic;

}

public boolean isAnaglyph() {

return isAnaglyph;

}

public void toggleBlending() {

if (this.isBlendingEnabled && !this.isBlendingForced){

makeToastText("Blending forced", Toast.LENGTH\_SHORT);

this.isBlendingEnabled = true;

this.isBlendingForced = true;

} else if (this.isBlendingForced){

makeToastText("Blending disabled", Toast.LENGTH\_SHORT);

this.isBlendingEnabled = false;

this.isBlendingForced = false;

} else {

makeToastText("Blending enabled", Toast.LENGTH\_SHORT);

this.isBlendingEnabled = true;

this.isBlendingForced = false;

}

}

public boolean isBlendingEnabled() {

return isBlendingEnabled;

}

public boolean isBlendingForced() {

return isBlendingForced;

}

@Override

public void onStart(){

ContentUtils.setThreadActivity(parent);

}

@Override

public void onLoadComplete(List<Object3DData> datas) {

// TODO: move texture load to LoaderTask

for (Object3DData data : datas) {

if (data.getTextureData() == null && data.getTextureFile() != null) {

Log.i("LoaderTask","Loading texture... "+data.getTextureFile());

try (InputStream stream = ContentUtils.getInputStream(data.getTextureFile())){

if (stream != null) {

data.setTextureData(IOUtils.read(stream));

}

} catch (IOException ex) {

data.addError("Problem loading texture " + data.getTextureFile());

}

}

}

// TODO: move error alert to LoaderTask

List<String> allErrors = new ArrayList<>();

for (Object3DData data : datas) {

addObject(data);

allErrors.addAll(data.getErrors());

}

if (!allErrors.isEmpty()){

makeToastText(allErrors.toString(), Toast.LENGTH\_LONG);

}

final String elapsed = (SystemClock.uptimeMillis() - startTime) / 1000 + " secs";

makeToastText("Build complete (" + elapsed + ")", Toast.LENGTH\_LONG);

ContentUtils.setThreadActivity(null);

}

@Override

public void onLoadError(Exception ex) {

Log.e("SceneLoader", ex.getMessage(), ex);

makeToastText("There was a problem building the model: " + ex.getMessage(), Toast.LENGTH\_LONG);

ContentUtils.setThreadActivity(null);

}

public Object3DData getSelectedObject() {

return selectedObject;

}

private void setSelectedObject(Object3DData selectedObject) {

this.selectedObject = selectedObject;

}

public void loadTexture(Object3DData obj, Uri uri) throws IOException {

if (obj == null && objects.size() != 1) {

makeToastText("Unavailable", Toast.LENGTH\_SHORT);

return;

}

obj = obj != null ? obj : objects.get(0);

obj.setTextureData(IOUtils.read(ContentUtils.getInputStream(uri)));

this.drawTextures = true;

}

public void processTouch(float x, float y) {

ModelRenderer mr = parent.getGLView().getModelRenderer();

Object3DData objectToSelect = CollisionDetection.getBoxIntersection(getObjects(), mr.getWidth(), mr.getHeight

(), mr.getModelViewMatrix(), mr.getModelProjectionMatrix(), x, y);

if (objectToSelect != null) {

if (getSelectedObject() == objectToSelect) {

Log.i("SceneLoader", "Unselected object " + objectToSelect.getId());

setSelectedObject(null);

} else {

Log.i("SceneLoader", "Selected object " + objectToSelect.getId());

setSelectedObject(objectToSelect);

}

if (isCollision()) {

Log.d("SceneLoader", "Detecting collision...");

float[] point = CollisionDetection.getTriangleIntersection(getObjects(), mr.getWidth(), mr.getHeight

(), mr.getModelViewMatrix(), mr.getModelProjectionMatrix(), x, y);

if (point != null) {

Log.i("SceneLoader", "Drawing intersection point: " + Arrays.toString(point));

addObject(Object3DBuilder.buildPoint(point).setColor(new float[]{1.0f, 0f, 0f, 1f}));

}

}

}

}

public void processMove(float dx1, float dy1) {

userHasInteracted = true;

}

public boolean isRotatingLight() {

return rotatingLight;

}

}

**android-3D-model-viewer/app/src/main/java/org/andresoviedo/app/model3D/demo/ExampleSceneLoader.java**

package org.andresoviedo.app.model3D.demo;

import android.annotation.SuppressLint;

import android.app.ProgressDialog;

import android.net.Uri;

import android.os.AsyncTask;

import android.util.Log;

import android.widget.Toast;

import org.andresoviedo.android\_3d\_model\_engine.model.Object3DData;

import org.andresoviedo.android\_3d\_model\_engine.services.Object3DBuilder;

import org.andresoviedo.app.model3D.view.ModelActivity;

import org.andresoviedo.util.android.ContentUtils;

import org.andresoviedo.util.io.IOUtils;

import java.io.InputStream;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* This class loads a 3D scene as an example of what can be done with the app

\*

\* @author andresoviedo

\*

\*/

public class ExampleSceneLoader extends SceneLoader {

public ExampleSceneLoader(ModelActivity modelActivity) {

super(modelActivity);

}

// TODO: fix this warning

@SuppressLint("StaticFieldLeak")

public void init() {

super.init();

new AsyncTask<Void, Void, Void>() {

ProgressDialog dialog = new ProgressDialog(parent);

List<Exception> errors = new ArrayList<>();

@Override

protected void onPreExecute() {

super.onPreExecute();

dialog.setCancelable(false);

dialog.setMessage("Loading demo...");

dialog.show();

}

@Override

protected Void doInBackground(Void... params) {

try {

// 3D Axis

setDrawAxis(true);

// test cube made of arrays

Object3DData obj10 = Object3DBuilder.buildCubeV1();

obj10.setColor(new float[] { 1f, 0f, 0f, 0.5f });

obj10.setPosition(new float[] { -2f, 2f, 0f });

addObject(obj10);

// test cube made of wires (I explode it to see the faces better)

Object3DData obj11 = Object3DBuilder.buildCubeV1();

obj11.setColor(new float[] { 1f, 1f, 0f, 0.5f });

obj11.setPosition(new float[] { 0f, 2f, 0f });

obj11.centerAndScaleAndExplode(1.0f, 1.5f);

obj11.setId(obj11.getId() + "\_exploded");

addObject(obj11);

// test cube made of wires (I explode it to see the faces better)

Object3DData obj12 = Object3DBuilder.buildCubeV1\_with\_normals();

obj12.setColor(new float[] { 1f, 0f, 1f, 1f });

obj12.setPosition(new float[] { 0f, 0f, -2f });

addObject(obj12);

// Set up ContentUtils so referenced materials and/or textures could be find

ContentUtils.setThreadActivity(parent);

ContentUtils.provideAssets(parent);

// test loading object

/\*try {

// this has no color array

Object3DData android = Object3DBuilder.loadV5(parent, Uri.parse("assets://assets/models/android.obj"));

android.setPosition(new float[] { 0f, 0f, 0f });

android.setColor(new float[] { 1.0f, 1.0f, 1.0f, 1.0f });

addObject(android);

} catch (Exception ex) {

errors.add(ex);

}\*/

// test cube made of indices

Object3DData obj20 = Object3DBuilder.buildSquareV2();

obj20.setColor(new float[] { 0f, 1f, 0, 0.25f });

obj20.setPosition(new float[] { 2f, 2f, 0f });

addObject(obj20);

// test cube with texture

try {

InputStream open = ContentUtils.getInputStream(Uri.parse("assets://assets/models/penguin.bmp"));

Object3DData obj3 = Object3DBuilder.buildCubeV3(IOUtils.read(open));

open.close();

obj3.setColor(new float[] { 1f, 1f, 1f, 1f });

obj3.setPosition(new float[] { -2f, -2f, 0f });

addObject(obj3);

} catch (Exception ex) {

errors.add(ex);

}

// test cube with texture & colors

try {

InputStream open = ContentUtils.getInputStream(Uri.parse("assets://assets/models/cube.bmp"));

Object3DData obj4 = Object3DBuilder.buildCubeV4(IOUtils.read(open));

open.close();

obj4.setColor(new float[] { 1f, 1f, 1f, 1f });

obj4.setPosition(new float[] { 0f, -2f, 0f });

addObject(obj4);

} catch (Exception ex) {

errors.add(ex);

}

// test loading object

try {

// this has no color array

Object3DData obj51 = Object3DBuilder.loadV5(parent, Uri.parse("assets://assets/models/teapot.obj"));

obj51.setPosition(new float[] { -2f, 0f, 0f });

obj51.setColor(new float[] { 1.0f, 1.0f, 0f, 1.0f });

addObject(obj51);

} catch (Exception ex) {

errors.add(ex);

}

// test loading object with materials

try {

// this has color array

Object3DData obj52 = Object3DBuilder.loadV5(parent, Uri.parse("assets://assets/models/cube.obj"));

obj52.setPosition(new float[] { 2f, -2f, 0f });

obj52.setColor(new float[] { 0.0f, 1.0f, 1f, 1.0f });

addObject(obj52);

} catch (Exception ex) {

errors.add(ex);

}

// test loading object made of polygonal faces

try {

// this has heterogeneous faces

Object3DData obj53 = Object3DBuilder.loadV5(parent, Uri.parse("assets://assets/models/ToyPlane.obj"));

InputStream open = ContentUtils.getInputStream(Uri.parse("assets://assets/models/"+obj53.getTextureFile()));

obj53.setTextureData(IOUtils.read(open));

obj53.centerAndScale(2.0f);

obj53.setPosition(new float[] { 2f, 0f, 0f });

obj53.setColor(new float[] { 1.0f, 1.0f, 1f, 1.0f });

// obj53.setDrawMode(GLES20.GL\_TRIANGLE\_FAN);

addObject(obj53);

} catch (Exception ex) {

errors.add(ex);

}

// test loading object without normals

/\*try {

Object3DData obj = Object3DBuilder.loadV5(parent, Uri.parse("assets://assets/models/cube4.obj"));

obj.setPosition(new float[] { 0f, 2f, -2f });

obj.setColor(new float[] { 0.3f, 0.52f, 1f, 1.0f });

addObject(obj);

} catch (Exception ex) {

errors.add(ex);

}\*/

} catch (Exception ex) {

errors.add(ex);

} finally{

ContentUtils.setThreadActivity(null);

ContentUtils.clearDocumentsProvided();

}

return null;

}

@Override

protected void onPostExecute(Void result) {

super.onPostExecute(result);

if (dialog.isShowing()) {

dialog.dismiss();

}

if (!errors.isEmpty()) {

StringBuilder msg = new StringBuilder("There was a problem loading the data");

for (Exception error : errors) {

Log.e("Example", error.getMessage(), error);

msg.append("\n" + error.getMessage());

}

Toast.makeText(parent.getApplicationContext(), msg, Toast.LENGTH\_LONG).show();

}

}

}.execute();

// test loading collada object

/\*try {

// this has heterogeneous faces

new ColladaLoaderTask(parent, Uri.parse("assets://assets/models/cowboy.dae"), this).execute();

} catch (Exception ex) {

Log.e("Example",ex.getMessage(),ex);

//errors.add(ex);

} finally {

ContentUtils.setThreadActivity(null);

ContentUtils.clearDocumentsProvided();

}\*/

}

}

**android-3D-model-viewer/app/src/main/java/org/andresoviedo/app/model3D/controller/TouchController.java /**

package org.andresoviedo.app.model3D.controller;

import android.graphics.PointF;

import android.opengl.Matrix;

import android.os.SystemClock;

import android.util.Log;

import android.view.MotionEvent;

import android.view.View;

import android.widget.ImageView;

import org.andresoviedo.android\_3d\_model\_engine.model.Camera;

import org.andresoviedo.app.model3D.demo.SceneLoader;

import org.andresoviedo.app.model3D.view.ModelRenderer;

import org.andresoviedo.app.model3D.view.ModelSurfaceView;

public class TouchController {

private static final String TAG = TouchController.class.getName();

private static final int TOUCH\_STATUS\_ZOOMING\_CAMERA = 1;

private static final int TOUCH\_STATUS\_ROTATING\_CAMERA = 4;

private static final int TOUCH\_STATUS\_MOVING\_WORLD = 5;

private final ModelSurfaceView view;

private final ModelRenderer mRenderer;

private int pointerCount = 0;

private float x1 = Float.MIN\_VALUE;

private float y1 = Float.MIN\_VALUE;

private float x2 = Float.MIN\_VALUE;

private float y2 = Float.MIN\_VALUE;

private float dx1 = Float.MIN\_VALUE;

private float dy1 = Float.MIN\_VALUE;

private float dx2 = Float.MIN\_VALUE;

private float dy2 = Float.MIN\_VALUE;

private float length = Float.MIN\_VALUE;

private float previousLength = Float.MIN\_VALUE;

private float currentPress1 = Float.MIN\_VALUE;

private float currentPress2 = Float.MIN\_VALUE;

private float rotation = 0;

private int currentSquare = Integer.MIN\_VALUE;

private boolean isOneFixedAndOneMoving = false;

private boolean fingersAreClosing = false;

private boolean isRotating = false;

private boolean gestureChanged = false;

private boolean moving = false;

private boolean simpleTouch = false;

private long lastActionTime;

private int touchDelay = -2;

private int touchStatus = -1;

private float previousX1;

private float previousY1;

private float previousX2;

private float previousY2;

private float[] previousVector = new float[4];

private float[] vector = new float[4];

private float[] rotationVector = new float[4];

private float previousRotationSquare;

public TouchController(ModelSurfaceView view, ModelRenderer renderer) {

super();

this.view = view;

this.mRenderer = renderer;

}

public synchronized boolean onTouchEvent(MotionEvent motionEvent) {

// MotionEvent reports input details from the touch screen

// and other input controls. In this case, you are only

// interested in events where the touch position changed.

switch (motionEvent.getActionMasked()) {

case MotionEvent.ACTION\_UP:

case MotionEvent.ACTION\_CANCEL:

case MotionEvent.ACTION\_POINTER\_UP:

case MotionEvent.ACTION\_HOVER\_EXIT:

case MotionEvent.ACTION\_OUTSIDE:

// this to handle "1 simple touch"

if (lastActionTime > SystemClock.uptimeMillis() - 250) {

simpleTouch = true;

} else {

gestureChanged = true;

touchDelay = 0;

lastActionTime = SystemClock.uptimeMillis();

simpleTouch = false;

}

moving = false;

break;

case MotionEvent.ACTION\_DOWN:

case MotionEvent.ACTION\_POINTER\_DOWN:

case MotionEvent.ACTION\_HOVER\_ENTER:

Log.v(TAG, "Gesture changed...");

gestureChanged = true;

touchDelay = 0;

lastActionTime = SystemClock.uptimeMillis();

simpleTouch = false;

break;

case MotionEvent.ACTION\_MOVE:

moving = true;

simpleTouch = false;

touchDelay++;

break;

default:

Log.w(TAG, "Unknown state: " + motionEvent.getAction());

gestureChanged = true;

}

pointerCount = motionEvent.getPointerCount();

if (pointerCount == 1) {

x1 = motionEvent.getX();

y1 = motionEvent.getY();

if (gestureChanged) {

Log.v(TAG, "x:" + x1 + ",y:" + y1);

previousX1 = x1;

previousY1 = y1;

}

dx1 = x1 - previousX1;

dy1 = y1 - previousY1;

} else if (pointerCount == 2) {

x1 = motionEvent.getX(0);

y1 = motionEvent.getY(0);

x2 = motionEvent.getX(1);

y2 = motionEvent.getY(1);

vector[0] = x2 - x1;

vector[1] = y2 - y1;

vector[2] = 0;

vector[3] = 1;

float len = Matrix.length(vector[0], vector[1], vector[2]);

vector[0] /= len;

vector[1] /= len;

// Log.v(TAG, "x1:" + x1 + ",y1:" + y1 + ",x2:" + x2 + ",y2:" + y2);

if (gestureChanged) {

previousX1 = x1;

previousY1 = y1;

previousX2 = x2;

previousY2 = y2;

System.arraycopy(vector, 0, previousVector, 0, vector.length);

}

dx1 = x1 - previousX1;

dy1 = y1 - previousY1;

dx2 = x2 - previousX2;

dy2 = y2 - previousY2;

rotationVector[0] = (previousVector[1] \* vector[2]) - (previousVector[2] \* vector[1]);

rotationVector[1] = (previousVector[2] \* vector[0]) - (previousVector[0] \* vector[2]);

rotationVector[2] = (previousVector[0] \* vector[1]) - (previousVector[1] \* vector[0]);

len = Matrix.length(rotationVector[0], rotationVector[1], rotationVector[2]);

rotationVector[0] /= len;

rotationVector[1] /= len;

rotationVector[2] /= len;

previousLength = (float) Math

.sqrt(Math.pow(previousX2 - previousX1, 2) + Math.pow(previousY2 - previousY1, 2));

length = (float) Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));

currentPress1 = motionEvent.getPressure(0);

currentPress2 = motionEvent.getPressure(1);

rotation = 0;

rotation = TouchScreen.getRotation360(motionEvent);

currentSquare = TouchScreen.getSquare(motionEvent);

if (currentSquare == 1 && previousRotationSquare == 4) {

rotation = 0;

} else if (currentSquare == 4 && previousRotationSquare == 1) {

rotation = 360;

}

// gesture detection

isOneFixedAndOneMoving = ((dx1 + dy1) == 0) != (((dx2 + dy2) == 0));

fingersAreClosing = !isOneFixedAndOneMoving && (Math.abs(dx1 + dx2) < 10 && Math.abs(dy1 + dy2) < 10);

isRotating = !isOneFixedAndOneMoving && (dx1 != 0 && dy1 != 0 && dx2 != 0 && dy2 != 0)

&& rotationVector[2] != 0;

}

if (pointerCount == 1 && simpleTouch) {

SceneLoader scene = view.getModelActivity().getScene();

scene.processTouch(x1,y1);

}

int max = Math.max(mRenderer.getWidth(), mRenderer.getHeight());

if (touchDelay > 1) {

// INFO: Process gesture

SceneLoader scene = view.getModelActivity().getScene();

scene.processMove(dx1, dy1);

Camera camera = scene.getCamera();

if (pointerCount == 1 && currentPress1 > 4.0f) {

} else if (pointerCount == 1) {

touchStatus = TOUCH\_STATUS\_MOVING\_WORLD;

// Log.v(TAG, "Translating camera (dx,dy) '" + dx1 + "','" + dy1 + "'...");

dx1 = (float)(dx1 / max \* Math.PI \* 2);

dy1 = (float)(dy1 / max \* Math.PI \* 2);

camera.translateCamera(dx1,dy1);

} else if (pointerCount == 2) {

if (fingersAreClosing) {

touchStatus = TOUCH\_STATUS\_ZOOMING\_CAMERA;

float zoomFactor = (length - previousLength) / max \* mRenderer.getFar();

Log.v(TAG, "Zooming '" + zoomFactor + "'...");

camera.MoveCameraZ(zoomFactor);

}

if (isRotating) {

touchStatus = TOUCH\_STATUS\_ROTATING\_CAMERA;

Log.v(TAG, "Rotating camera '" + Math.signum(rotationVector[2]) + "'...");

camera.Rotate((float) (Math.signum(rotationVector[2]) / Math.PI) / 4);

}

}

// INFO: Realizamos la acci�n

// switch (touchStatus) {

// case TOUCH\_STATUS\_ROTATING\_OBJECT:

// // reverse direction of rotation above the mid-line

// if (y > getHeight() / 2) {

// // Log.d(TAG, "Reversing dx");

// dx = dx \* -1;

// }

//

// // reverse direction of rotation to left of the mid-line

// if (x < getWidth() / 2) {

// // Log.d(TAG, "Reversing dy");

// dy = dy \* -1;

// }

// Log.w(TAG, "Rotating '" + dx + "','" + dy + "'...");

//

// if (wzSquare > wzTriangle) {

// mRenderer.getmSquare().setRotationZ(mRenderer.getmSquare().getRotationZ() + ((dx + dy) \*

// TOUCH\_SCALE\_FACTOR));

// } else if (wzTriangle > wzSquare) {

// mRenderer.getmTriangle().setRotationZ(mRenderer.getmTriangle().getRotationZ() + ((dx + dy) \*

// TOUCH\_SCALE\_FACTOR));

// }

//

// break;

//

// case TOUCH\_STATUS\_MOVING\_OBJECT:

// // TODO: guess front object

// boolean sqHit = wzSquare > wzTriangle;

// boolean triHit = wzTriangle > wzSquare;

// Log.w(TAG, "Moving '" + sqHit + "','" + triHit + "'...");

//

// if (sqHit) {

// mRenderer.getmSquare().translateX((dx \* mRenderer.getRatio() \* 2 / mRenderer.getWidth()) \*

// TOUCH\_MOVE\_FACTOR);

// mRenderer.getmSquare().translateY((-dy \* 1 / mRenderer.getHeight()) \* TOUCH\_MOVE\_FACTOR);

// }

// if (triHit) {

// mRenderer.getmTriangle().translateX((dx \* mRenderer.getRatio() \* 2 / mRenderer.getWidth()) \*

// TOUCH\_MOVE\_FACTOR);

// mRenderer.getmTriangle().translateY((-dy \* 1 / mRenderer.getHeight()) \* TOUCH\_MOVE\_FACTOR);

// }

// break;

//

// case TOUCH\_STATUS\_ROTATING\_OBJECT2:

// Log.w(TAG, "Rotating '" + wzSquare + "','" + wzTriangle + "'...");

// // INFO: We are moving 2 fingers in different directions

// // Rotate Camera

// // TODO: Rotationfactor deber�a ser proporcional a la z?

// if (wzSquare > wzTriangle) {

// mRenderer.getmSquare().setRotationZ(mRenderer.getmSquare().getRotationZ() + (actualRotation \*

// TOUCH\_ROTATION\_FACTOR));

// } else if (wzTriangle > wzSquare) {

// mRenderer.getmTriangle().setRotationZ(mRenderer.getmTriangle().getRotationZ() + (actualRotation \*

// TOUCH\_ROTATION\_FACTOR));

// }

// // mRenderer.setAngle(mRenderer.getAngle() + actualRotation

// // \* TOUCH\_ROTATION\_FACTOR);

//

// // mRenderer.getCamera().Rotate(rotation \*

// // CAMERA\_ROTATION\_FACTOR,

// // 0);

//

// break;

//}

}

previousX1 = x1;

previousY1 = y1;

previousX2 = x2;

previousY2 = y2;

previousRotationSquare = currentSquare;

System.arraycopy(vector, 0, previousVector, 0, vector.length);

if (gestureChanged && touchDelay > 1) {

gestureChanged = false;

Log.v(TAG, "Fin");

}

view.requestRender();

return true;

}

}

class TouchScreen {

// these matrices will be used to move and zoom image

private android.graphics.Matrix matrix = new android.graphics.Matrix();

private android.graphics.Matrix savedMatrix = new android.graphics.Matrix();

// we can be in one of these 3 states

private static final int NONE = 0;

private static final int DRAG = 1;

private static final int ZOOM = 2;

private int mode = NONE;

// remember some things for zooming

private PointF start = new PointF();

private PointF mid = new PointF();

private float oldDist = 1f;

private float d = 0f;

private float newRot = 0f;

private float[] lastEvent = null;

public boolean onTouch(View v, MotionEvent event) {

// handle touch events here

ImageView view = (ImageView) v;

switch (event.getAction() & MotionEvent.ACTION\_MASK) {

case MotionEvent.ACTION\_DOWN:

savedMatrix.set(matrix);

start.set(event.getX(), event.getY());

mode = DRAG;

lastEvent = null;

break;

case MotionEvent.ACTION\_POINTER\_DOWN:

oldDist = spacing(event);

if (oldDist > 10f) {

savedMatrix.set(matrix);

midPoint(mid, event);

mode = ZOOM;

}

lastEvent = new float[4];

lastEvent[0] = event.getX(0);

lastEvent[1] = event.getX(1);

lastEvent[2] = event.getY(0);

lastEvent[3] = event.getY(1);

d = getRotation(event);

break;

case MotionEvent.ACTION\_UP:

case MotionEvent.ACTION\_POINTER\_UP:

mode = NONE;

lastEvent = null;

break;

case MotionEvent.ACTION\_MOVE:

if (mode == DRAG) {

matrix.set(savedMatrix);

float dx = event.getX() - start.x;

float dy = event.getY() - start.y;

matrix.postTranslate(dx, dy);

} else if (mode == ZOOM) {

float newDist = spacing(event);

if (newDist > 10f) {

matrix.set(savedMatrix);

float scale = (newDist / oldDist);

matrix.postScale(scale, scale, mid.x, mid.y);

}

if (lastEvent != null && event.getPointerCount() == 3) {

newRot = getRotation(event);

float r = newRot - d;

float[] values = new float[9];

matrix.getValues(values);

float tx = values[2];

float ty = values[5];

float sx = values[0];

float xc = (view.getWidth() / 2) \* sx;

float yc = (view.getHeight() / 2) \* sx;

matrix.postRotate(r, tx + xc, ty + yc);

}

}

break;

}

view.setImageMatrix(matrix);

return true;

}

/\*\*

\* Determine the space between the first two fingers

\*/

private float spacing(MotionEvent event) {

float x = event.getX(0) - event.getX(1);

float y = event.getY(0) - event.getY(1);

return (float)Math.sqrt(x \* x + y \* y);

}

/\*\*

\* Calculate the mid point of the first two fingers

\*/

private void midPoint(PointF point, MotionEvent event) {

float x = event.getX(0) + event.getX(1);

float y = event.getY(0) + event.getY(1);

point.set(x / 2, y / 2);

}

/\*\*

\* Calculate the degree to be rotated by.

\*

\* @param event

\* @return Degrees

\*/

public static float getRotation(MotionEvent event) {

double dx = (event.getX(0) - event.getX(1));

double dy = (event.getY(0) - event.getY(1));

double radians = Math.atan2(Math.abs(dy), Math.abs(dx));

double degrees = Math.toDegrees(radians);

return (float) degrees;

}

public static float getRotation360(MotionEvent event) {

double dx = (event.getX(0) - event.getX(1));

double dy = (event.getY(0) - event.getY(1));

double radians = Math.atan2(Math.abs(dy), Math.abs(dx));

double degrees = Math.toDegrees(radians);

int square = 1;

if (dx > 0 && dy == 0) {

square = 1;

} else if (dx > 0 && dy < 0) {

square = 1;

} else if (dx == 0 && dy < 0) {

square = 2;

degrees = 180 - degrees;

} else if (dx < 0 && dy < 0) {

square = 2;

degrees = 180 - degrees;

} else if (dx < 0 && dy == 0) {

square = 3;

degrees = 180 + degrees;

} else if (dx < 0 && dy > 0) {

square = 3;

degrees = 180 + degrees;

} else if (dx == 0 && dy > 0) {

square = 4;

degrees = 360 - degrees;

} else if (dx > 0 && dy > 0) {

square = 4;

degrees = 360 - degrees;

}

return (float) degrees;

}

public static int getSquare(MotionEvent event) {

double dx = (event.getX(0) - event.getX(1));

double dy = (event.getY(0) - event.getY(1));

int square = 1;

if (dx > 0 && dy == 0) {

square = 1;

} else if (dx > 0 && dy < 0) {

square = 1;

} else if (dx == 0 && dy < 0) {

square = 2;

} else if (dx < 0 && dy < 0) {

square = 2;

} else if (dx < 0 && dy == 0) {

square = 3;

} else if (dx < 0 && dy > 0) {

square = 3;

} else if (dx == 0 && dy > 0) {

square = 4;

} else if (dx > 0 && dy > 0) {

square = 4;

}

return square;

}

}

**package org.andresoviedo.app.model3D.controller;**

import android.graphics.PointF;

import android.opengl.Matrix;

import android.os.SystemClock;

import android.util.Log;

import android.view.MotionEvent;

import android.view.View;

import android.widget.ImageView;

import org.andresoviedo.android\_3d\_model\_engine.model.Camera;

import org.andresoviedo.app.model3D.demo.SceneLoader;

import org.andresoviedo.app.model3D.view.ModelRenderer;

import org.andresoviedo.app.model3D.view.ModelSurfaceView;

public class TouchController {

private static final String TAG = TouchController.class.getName();

private static final int TOUCH\_STATUS\_ZOOMING\_CAMERA = 1;

private static final int TOUCH\_STATUS\_ROTATING\_CAMERA = 4;

private static final int TOUCH\_STATUS\_MOVING\_WORLD = 5;

private final ModelSurfaceView view;

private final ModelRenderer mRenderer;

private int pointerCount = 0;

private float x1 = Float.MIN\_VALUE;

private float y1 = Float.MIN\_VALUE;

private float x2 = Float.MIN\_VALUE;

private float y2 = Float.MIN\_VALUE;

private float dx1 = Float.MIN\_VALUE;

private float dy1 = Float.MIN\_VALUE;

private float dx2 = Float.MIN\_VALUE;

private float dy2 = Float.MIN\_VALUE;

private float length = Float.MIN\_VALUE;

private float previousLength = Float.MIN\_VALUE;

private float currentPress1 = Float.MIN\_VALUE;

private float currentPress2 = Float.MIN\_VALUE;

private float rotation = 0;

private int currentSquare = Integer.MIN\_VALUE;

private boolean isOneFixedAndOneMoving = false;

private boolean fingersAreClosing = false;

private boolean isRotating = false;

private boolean gestureChanged = false;

private boolean moving = false;

private boolean simpleTouch = false;

private long lastActionTime;

private int touchDelay = -2;

private int touchStatus = -1;

private float previousX1;

private float previousY1;

private float previousX2;

private float previousY2;

private float[] previousVector = new float[4];

private float[] vector = new float[4];

private float[] rotationVector = new float[4];

private float previousRotationSquare;

public TouchController(ModelSurfaceView view, ModelRenderer renderer) {

super();

this.view = view;

this.mRenderer = renderer;

}

public synchronized boolean onTouchEvent(MotionEvent motionEvent) {

// MotionEvent reports input details from the touch screen

// and other input controls. In this case, you are only

// interested in events where the touch position changed.

switch (motionEvent.getActionMasked()) {

case MotionEvent.ACTION\_UP:

case MotionEvent.ACTION\_CANCEL:

case MotionEvent.ACTION\_POINTER\_UP:

case MotionEvent.ACTION\_HOVER\_EXIT:

case MotionEvent.ACTION\_OUTSIDE:

// this to handle "1 simple touch"

if (lastActionTime > SystemClock.uptimeMillis() - 250) {

simpleTouch = true;

} else {

gestureChanged = true;

touchDelay = 0;

lastActionTime = SystemClock.uptimeMillis();

simpleTouch = false;

}

moving = false;

break;

case MotionEvent.ACTION\_DOWN:

case MotionEvent.ACTION\_POINTER\_DOWN:

case MotionEvent.ACTION\_HOVER\_ENTER:

Log.v(TAG, "Gesture changed...");

gestureChanged = true;

touchDelay = 0;

lastActionTime = SystemClock.uptimeMillis();

simpleTouch = false;

break;

case MotionEvent.ACTION\_MOVE:

moving = true;

simpleTouch = false;

touchDelay++;

break;

default:

Log.w(TAG, "Unknown state: " + motionEvent.getAction());

gestureChanged = true;

}

pointerCount = motionEvent.getPointerCount();

if (pointerCount == 1) {

x1 = motionEvent.getX();

y1 = motionEvent.getY();

if (gestureChanged) {

Log.v(TAG, "x:" + x1 + ",y:" + y1);

previousX1 = x1;

previousY1 = y1;

}

dx1 = x1 - previousX1;

dy1 = y1 - previousY1;

} else if (pointerCount == 2) {

x1 = motionEvent.getX(0);

y1 = motionEvent.getY(0);

x2 = motionEvent.getX(1);

y2 = motionEvent.getY(1);

vector[0] = x2 - x1;

vector[1] = y2 - y1;

vector[2] = 0;

vector[3] = 1;

float len = Matrix.length(vector[0], vector[1], vector[2]);

vector[0] /= len;

vector[1] /= len;

// Log.v(TAG, "x1:" + x1 + ",y1:" + y1 + ",x2:" + x2 + ",y2:" + y2);

if (gestureChanged) {

previousX1 = x1;

previousY1 = y1;

previousX2 = x2;

previousY2 = y2;

System.arraycopy(vector, 0, previousVector, 0, vector.length);

}

dx1 = x1 - previousX1;

dy1 = y1 - previousY1;

dx2 = x2 - previousX2;

dy2 = y2 - previousY2;

rotationVector[0] = (previousVector[1] \* vector[2]) - (previousVector[2] \* vector[1]);

rotationVector[1] = (previousVector[2] \* vector[0]) - (previousVector[0] \* vector[2]);

rotationVector[2] = (previousVector[0] \* vector[1]) - (previousVector[1] \* vector[0]);

len = Matrix.length(rotationVector[0], rotationVector[1], rotationVector[2]);

rotationVector[0] /= len;

rotationVector[1] /= len;

rotationVector[2] /= len;

previousLength = (float) Math

.sqrt(Math.pow(previousX2 - previousX1, 2) + Math.pow(previousY2 - previousY1, 2));

length = (float) Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));

currentPress1 = motionEvent.getPressure(0);

currentPress2 = motionEvent.getPressure(1);

rotation = 0;

rotation = TouchScreen.getRotation360(motionEvent);

currentSquare = TouchScreen.getSquare(motionEvent);

if (currentSquare == 1 && previousRotationSquare == 4) {

rotation = 0;

} else if (currentSquare == 4 && previousRotationSquare == 1) {

rotation = 360;

}

// gesture detection

isOneFixedAndOneMoving = ((dx1 + dy1) == 0) != (((dx2 + dy2) == 0));

fingersAreClosing = !isOneFixedAndOneMoving && (Math.abs(dx1 + dx2) < 10 && Math.abs(dy1 + dy2) < 10);

isRotating = !isOneFixedAndOneMoving && (dx1 != 0 && dy1 != 0 && dx2 != 0 && dy2 != 0)

&& rotationVector[2] != 0;

}

if (pointerCount == 1 && simpleTouch) {

SceneLoader scene = view.getModelActivity().getScene();

scene.processTouch(x1,y1);

}

int max = Math.max(mRenderer.getWidth(), mRenderer.getHeight());

if (touchDelay > 1) {

// INFO: Process gesture

SceneLoader scene = view.getModelActivity().getScene();

scene.processMove(dx1, dy1);

Camera camera = scene.getCamera();

if (pointerCount == 1 && currentPress1 > 4.0f) {

} else if (pointerCount == 1) {

touchStatus = TOUCH\_STATUS\_MOVING\_WORLD;

// Log.v(TAG, "Translating camera (dx,dy) '" + dx1 + "','" + dy1 + "'...");

dx1 = (float)(dx1 / max \* Math.PI \* 2);

dy1 = (float)(dy1 / max \* Math.PI \* 2);

camera.translateCamera(dx1,dy1);

} else if (pointerCount == 2) {

if (fingersAreClosing) {

touchStatus = TOUCH\_STATUS\_ZOOMING\_CAMERA;

float zoomFactor = (length - previousLength) / max \* mRenderer.getFar();

Log.v(TAG, "Zooming '" + zoomFactor + "'...");

camera.MoveCameraZ(zoomFactor);

}

if (isRotating) {

touchStatus = TOUCH\_STATUS\_ROTATING\_CAMERA;

Log.v(TAG, "Rotating camera '" + Math.signum(rotationVector[2]) + "'...");

camera.Rotate((float) (Math.signum(rotationVector[2]) / Math.PI) / 4);

}

}

// INFO: Realizamos la acci�n

// switch (touchStatus) {

// case TOUCH\_STATUS\_ROTATING\_OBJECT:

// // reverse direction of rotation above the mid-line

// if (y > getHeight() / 2) {

// // Log.d(TAG, "Reversing dx");

// dx = dx \* -1;

// }

//

// // reverse direction of rotation to left of the mid-line

// if (x < getWidth() / 2) {

// // Log.d(TAG, "Reversing dy");

// dy = dy \* -1;

// }

// Log.w(TAG, "Rotating '" + dx + "','" + dy + "'...");

//

// if (wzSquare > wzTriangle) {

// mRenderer.getmSquare().setRotationZ(mRenderer.getmSquare().getRotationZ() + ((dx + dy) \*

// TOUCH\_SCALE\_FACTOR));

// } else if (wzTriangle > wzSquare) {

// mRenderer.getmTriangle().setRotationZ(mRenderer.getmTriangle().getRotationZ() + ((dx + dy) \*

// TOUCH\_SCALE\_FACTOR));

// }

//

// break;

//

// case TOUCH\_STATUS\_MOVING\_OBJECT:

// // TODO: guess front object

// boolean sqHit = wzSquare > wzTriangle;

// boolean triHit = wzTriangle > wzSquare;

// Log.w(TAG, "Moving '" + sqHit + "','" + triHit + "'...");

//

// if (sqHit) {

// mRenderer.getmSquare().translateX((dx \* mRenderer.getRatio() \* 2 / mRenderer.getWidth()) \*

// TOUCH\_MOVE\_FACTOR);

// mRenderer.getmSquare().translateY((-dy \* 1 / mRenderer.getHeight()) \* TOUCH\_MOVE\_FACTOR);

// }

// if (triHit) {

// mRenderer.getmTriangle().translateX((dx \* mRenderer.getRatio() \* 2 / mRenderer.getWidth()) \*

// TOUCH\_MOVE\_FACTOR);

// mRenderer.getmTriangle().translateY((-dy \* 1 / mRenderer.getHeight()) \* TOUCH\_MOVE\_FACTOR);

// }

// break;

//

// case TOUCH\_STATUS\_ROTATING\_OBJECT2:

// Log.w(TAG, "Rotating '" + wzSquare + "','" + wzTriangle + "'...");

// // INFO: We are moving 2 fingers in different directions

// // Rotate Camera

// // TODO: Rotationfactor deber�a ser proporcional a la z?

// if (wzSquare > wzTriangle) {

// mRenderer.getmSquare().setRotationZ(mRenderer.getmSquare().getRotationZ() + (actualRotation \*

// TOUCH\_ROTATION\_FACTOR));

// } else if (wzTriangle > wzSquare) {

// mRenderer.getmTriangle().setRotationZ(mRenderer.getmTriangle().getRotationZ() + (actualRotation \*

// TOUCH\_ROTATION\_FACTOR));

// }

// // mRenderer.setAngle(mRenderer.getAngle() + actualRotation

// // \* TOUCH\_ROTATION\_FACTOR);

//

// // mRenderer.getCamera().Rotate(rotation \*

// // CAMERA\_ROTATION\_FACTOR,

// // 0);

//

// break;

//}

}

previousX1 = x1;

previousY1 = y1;

previousX2 = x2;

previousY2 = y2;

previousRotationSquare = currentSquare;

System.arraycopy(vector, 0, previousVector, 0, vector.length);

if (gestureChanged && touchDelay > 1) {

gestureChanged = false;

Log.v(TAG, "Fin");

}

view.requestRender();

return true;

}

}

class TouchScreen {

// these matrices will be used to move and zoom image

private android.graphics.Matrix matrix = new android.graphics.Matrix();

private android.graphics.Matrix savedMatrix = new android.graphics.Matrix();

// we can be in one of these 3 states

private static final int NONE = 0;

private static final int DRAG = 1;

private static final int ZOOM = 2;

private int mode = NONE;

// remember some things for zooming

private PointF start = new PointF();

private PointF mid = new PointF();

private float oldDist = 1f;

private float d = 0f;

private float newRot = 0f;

private float[] lastEvent = null;

public boolean onTouch(View v, MotionEvent event) {

// handle touch events here

ImageView view = (ImageView) v;

switch (event.getAction() & MotionEvent.ACTION\_MASK) {

case MotionEvent.ACTION\_DOWN:

savedMatrix.set(matrix);

start.set(event.getX(), event.getY());

mode = DRAG;

lastEvent = null;

break;

case MotionEvent.ACTION\_POINTER\_DOWN:

oldDist = spacing(event);

if (oldDist > 10f) {

savedMatrix.set(matrix);

midPoint(mid, event);

mode = ZOOM;

}

lastEvent = new float[4];

lastEvent[0] = event.getX(0);

lastEvent[1] = event.getX(1);

lastEvent[2] = event.getY(0);

lastEvent[3] = event.getY(1);

d = getRotation(event);

break;

case MotionEvent.ACTION\_UP:

case MotionEvent.ACTION\_POINTER\_UP:

mode = NONE;

lastEvent = null;

break;

case MotionEvent.ACTION\_MOVE:

if (mode == DRAG) {

matrix.set(savedMatrix);

float dx = event.getX() - start.x;

float dy = event.getY() - start.y;

matrix.postTranslate(dx, dy);

} else if (mode == ZOOM) {

float newDist = spacing(event);

if (newDist > 10f) {

matrix.set(savedMatrix);

float scale = (newDist / oldDist);

matrix.postScale(scale, scale, mid.x, mid.y);

}

if (lastEvent != null && event.getPointerCount() == 3) {

newRot = getRotation(event);

float r = newRot - d;

float[] values = new float[9];

matrix.getValues(values);

float tx = values[2];

float ty = values[5];

float sx = values[0];

float xc = (view.getWidth() / 2) \* sx;

float yc = (view.getHeight() / 2) \* sx;

matrix.postRotate(r, tx + xc, ty + yc);

}

}

break;

}

view.setImageMatrix(matrix);

return true;

}

/\*\*

\* Determine the space between the first two fingers

\*/

private float spacing(MotionEvent event) {

float x = event.getX(0) - event.getX(1);

float y = event.getY(0) - event.getY(1);

return (float)Math.sqrt(x \* x + y \* y);

}

/\*\*

\* Calculate the mid point of the first two fingers

\*/

private void midPoint(PointF point, MotionEvent event) {

float x = event.getX(0) + event.getX(1);

float y = event.getY(0) + event.getY(1);

point.set(x / 2, y / 2);

}

/\*\*

\* Calculate the degree to be rotated by.

\*

\* @param event

\* @return Degrees

\*/

public static float getRotation(MotionEvent event) {

double dx = (event.getX(0) - event.getX(1));

double dy = (event.getY(0) - event.getY(1));

double radians = Math.atan2(Math.abs(dy), Math.abs(dx));

double degrees = Math.toDegrees(radians);

return (float) degrees;

}

public static float getRotation360(MotionEvent event) {

double dx = (event.getX(0) - event.getX(1));

double dy = (event.getY(0) - event.getY(1));

double radians = Math.atan2(Math.abs(dy), Math.abs(dx));

double degrees = Math.toDegrees(radians);

int square = 1;

if (dx > 0 && dy == 0) {

square = 1;

} else if (dx > 0 && dy < 0) {

square = 1;

} else if (dx == 0 && dy < 0) {

square = 2;

degrees = 180 - degrees;

} else if (dx < 0 && dy < 0) {

square = 2;

degrees = 180 - degrees;

} else if (dx < 0 && dy == 0) {

square = 3;

degrees = 180 + degrees;

} else if (dx < 0 && dy > 0) {

square = 3;

degrees = 180 + degrees;

} else if (dx == 0 && dy > 0) {

square = 4;

degrees = 360 - degrees;

} else if (dx > 0 && dy > 0) {

square = 4;

degrees = 360 - degrees;

}

return (float) degrees;

}

public static int getSquare(MotionEvent event) {

double dx = (event.getX(0) - event.getX(1));

double dy = (event.getY(0) - event.getY(1));

int square = 1;

if (dx > 0 && dy == 0) {

square = 1;

} else if (dx > 0 && dy < 0) {

square = 1;

} else if (dx == 0 && dy < 0) {

square = 2;

} else if (dx < 0 && dy < 0) {

square = 2;

} else if (dx < 0 && dy == 0) {

square = 3;

} else if (dx < 0 && dy > 0) {

square = 3;

} else if (dx == 0 && dy > 0) {

square = 4;

} else if (dx > 0 && dy > 0) {

square = 4;

}

return square;

}

}

**android-3D-model-viewer/app/src/main/java/org/andresoviedo/util/view/TextActivity.java /**

package org.andresoviedo.util.view;

import android.app.Activity;

import android.os.Bundle;

import android.text.method.LinkMovementMethod;

import android.widget.TextView;

import org.andresoviedo.dddmodel2.R;

public class TextActivity extends Activity {

private TextView text;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_text);

Bundle b = getIntent().getExtras();

String title = b.getString("title");

setTitle(title);

String value = b.getString("text");

text = findViewById(R.id.text\_activity\_text);

text.setMovementMethod(LinkMovementMethod.getInstance());

text.setText(value);

}

}