Android Projects

COT 5930

Professor Ravi Shankar

Android and Animation

By Brian Chamba

11/28/2010Abstract

The purpose of this project is to integrate animation with the user interface of the Android chess game. The movement of the robots, whether moving from one square to another or getting out of the way so another robot can move, takes time. During this time, it makes sense to have some sort of animation on the screen of the phone which depicts what is going on and also prevents the user from trying to interact with the main chess screen. In order for this to be successful, different animations (taking an enemy piece, being captured, basic movement, etc) must be called at specific times. While these animations and the pieces performing them will most likely have to be developed by an artist, placeholder animations can be used in the meantime. The main goal is to be able to call upon a specific animation with a given command. More specifically, the tasks for this project are as follows:

1. Establish protocol for artists to export MD2 files from Blender. Blender will be used because it is free and open source. The models and animation files can be created here, while the textures for the models can be created in Gimp, another free and open source program.
2. Program the ability to add separate animations to be played.
3. Program the ability to control pieces individually.
4. Create function for camera to move in front of appropriate piece while animation occurs.

Background

The user interface for the chess game can easily implemented using a user interface I created for a 3x3 chess game. With this application, I was able to import 3d objects made in Maya as obj files into the chess game.

While the objects seen in the picture above can be rotated and translated in 3d space making for what seems like animation, they cannot do everything that can be done if the animations were created with an outside program, such as Blender. I have seen some libraries that can import files that contain animation, but they do not seem to be able to control the animation or determine when to start and stop it.

Methods

The first step in this project will be to find a way to import a file into Android that contains animation. Most likely a library that can do this already exists.

Next, the animation in the file should be controllable. This will require figuring out how to start and stop a specific animation that the object contains.

After this is done, the pieces and their animations will need to be implemented and tied in with the user interface for the chess game.

Results

The results obtained from this project show that the original tasks stated in the abstract were successfully reached. Since it has been decided that Blender was to be used to create 3d models and the animations for them, no lengthy method for exporting MD2 files from Blender was necessary to come up with. This is because Blender already contains an option for exporting

these type of files. All that is left for the artist creating the models then is to become familiar with Blender if they have not used this software previously.

The next tasks involved programming the ability to add animations to be played in an Android environment and then play them on command. The next few pictures convey this and the actual implementation will be explained in the discussion.

(normal game screen)

(bottom right piece selected)

(walking animation played for moving piece)

(attacking animation played when capturing enemy piece)

(falling animation played for captured piece)

Although the pictures are not capable of showing actual movement, it is clear to see that the model is performing different movements or animations.

Lastly, the pictures shown above also show that the camera is not stationary. The camera moves in front of the model that is performing either a walking, attacking, or falling animation. After this animation is complete, the camera returns to its original overhead position.

Discussion

The first step in this project required loading the MD2 files into objects that can be seen in a 3d environment on an Android device. To do this, I used a library(libgdx), created by badlogic games, that I had also used previously for loading the obj files. The loading of the model is done with the following lines of code:

```
InputStream in = activity.getAssets().open("knight.md2");
knight = MD2Loader.load( gl, in );
instanceKnight = new MD2Instance( knight, 0.05f );
```

instanceKnight now represents the 3d model seen in the pictures previously shown.

Next, animations needed to be added for playback. In this situation, the artist would normally provide the programmer with a text file containing the names of the animations and the frames during which they occur. For example, the text file might have:

Walking-0-10

Jumping-11-20

Kicking-21-30.….

However, since I did not work with the person who created this particular model, I had to do some guessing as to where the animations I needed occurred. With this information, I created a function for storing these animations. The function, contained in the original libraries MD2 instance class, is as below:

```
public void addAnimation(int animation,int start, int finish)
{
	this.animS.add(animation, start);
	this.animF.add(animation, finish);
}
```

Here, animS and animF are arrays that store all the starting and finishing frames for all the animations for one particular model type, in this case the knight.

The calls for this function look like the following lines:

```
instanceKnight.addAnimation(WALKING,40,45);
instanceKnight.addAnimation(WAVING, 70, 76);
instanceKnight.addAnimation(FALLING,170,197);
```

```
instanceKnight.addAnimation(ATTACKING, 150, 165);
```

The first variable being passed in the function call must be constant and unique. This assures that all animations get placed in the array in a position that can be recalled when wanting to play the animation.

Once all animations for a particular model are added, functions must be added to the class the model represents. In our example, there is only one class called checkerPiece. If this were a game of chess, there would be classed for each piece(pawn,rook,bishop…). The reason for storing the functions in this class, and not the MD2 instance class that was previously modified, is we want the animations to occur for particular pieces. The only way to make sure only the selected piece is playing the animation desired, is to place the play animation function in that particular piece's class. The functions are below and will be explained in further detail:

```
public void playAnimation(int animation,MD2Instance instance)
{
	this.begin_frame=instance.animS.get(animation);
	this.current_frame=this.begin_frame;
	this.end_frame=instance.animF.get(animation);
}

public void checkFrames(int delayTime)
{
	delay++;
	if(delay>delayTime)
		{
		this.current_frame++;
		delay=0;
		}
	if(this.current_frame>this.end_frame)
		this.current_frame=this.begin_frame;
}

public void endAnimation()
{
	this.begin_frame=this.current_frame=this.end_frame=0;
}
```

The MD2 instance class has a function that allows the selection for which frames to be played. Using this, in conjunction with the above functions, allows control of individual pieces and their animations.

The MD2 function being used is setFrameRange(int,int). This is called during the main game loop right before each individual piece is rendered and right after the above function checkFrames. Initially, each piece has it's current frame, begin frame, and end frame variables set to 0. CheckFrames makes sure that the current frame of the piece is not exceeding the end frame for the animation it is supposed to be displaying. Since the begin and end frames are normally at 0, when setFrameRange is called, the individual piece is constantly rendered in the same frame. However, when a piece receives the playAnimation call, the begin and end frames are set to the frames corresponding to the desired animation. Now, each time setFrameRange is called in the main loop, it is called with different variables. For instance, when a piece is selected, piece[k].playAnimation(WAVING, instanceKnight) is called. This sets the begin and end frames for this particular piece in the array of pieces to 70 and 76, respectively. During the main loop, the following calls are made:

SetFrameRange(70,70)…setFrameRange(71,71)…setFrameRange(72,72)…

This allows for particular frames to be cycled through for individual pieces.

When an animation is occurring that takes up the screen, such as the walking, attacking, and falling animations(seen in the previous photos), the camera needs to move. The moving of the camera is what probably took up the most time on this project. To do this, the GLU function gluLookAt was used. This function takes 10 parameters. The most important parameters are the x,y coordinates of the camera and the x,y coordinates of the spot the camera should look at. However, these coordinates do not behave how I assumed they would. For example, the top right piece has the coordinates 4,4. The z coordinate is of no importance. During experimentation, trying to get the camera to point at this piece, I found out that the coordinate for the piece ended up being 0.7,0.5. I am not sure why the coordinates are different since they are

in the same xyz plane. Whatever the reason, I had to come up with a way to look at the piece based on it's known xy position. For this, the gluLookAt function looks as such:

```
      GLU.gluLookAt(gl,animX/5f,(animY/8f)
+1.5f(Piece[animPlace].getRow()/10.0f), -0.2f, animX/5f, (animY/8f), 0, 0,
-1, 0);
```

Using the example from above, dividing the x,y coordinates by the known x,y coordinates of the top right piece, I found that the x position needs to be divided by 5 and the y position by 8 in order for the camera and point of interest to be in the correct location. The y position of the camera does need to be increased a bit so it can view the piece and not be inside of it.

Further experimentation has to be done with the camera, so when these functions are used outside of the checkers or chess application, where you may have objects randomly scattered about, they can still be used to point at objects given the xyz coordinates of it.

Conclusion

With the previously discussed functions added to the small scale checkers application, animations can now be played on command. From here, the main focus should be on creating the rest of the models, animations and textures for the models to be used in the full scale chess game. It is important to note that the game board, even though it probably will have no animation, still has to be an MD2 file, as the library does not allow for the importing of both MD2 and obj files at the same time.

Development also needs to occur for the chess game logic. This can stem from the small scale checkers application or can be developed from scratch. It may actually be easier to start from scratch as the checkers application can and should be vastly improved upon. Once the game logic is set up, the user interface can be incorporated by including the libgdx library and adding functions to the classes for the different pieces. Once again, more work needs to be done

with the camera. It works for now as it is, but the function used for it will change based on the size of the game board. A simpler way to look at an object given its xyz coordinates needs to be developed.

References

1. Libgdx library - http://www.badlogicgames.com/wordpress/
2. Android developer site - http://d.android.com/index.html
3. Blender 3D creation tool - http://www.blender.org/
4. Gimp image manipulation program - http://www.gimp.org/