CG Exercise #3 - Q*bert (OpenGL exercise)

Go to this exercise's FAQ and Updates page

Introduction

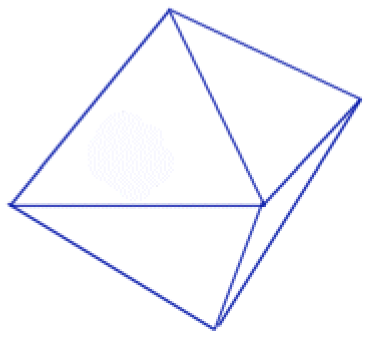
In this exercise you will implement a 3D version of the 1982 computer game called Q*bert. In the original Q*bert game the user plays a small ball like creature which hops around on a single side of an isometric pyramid structure made of cubes. When Q*bert hops on to a cube the cube changes color. The purpose of the game is to change the color of the all the cubes in the structure.
more about the original Q*bert game:
http://en.wikipedia.org/wiki/Q*bert
a flash implementation:
http://www.gamingdelight.com/games/qbert.php
The original game:
http://www.jrok.com/pcbert.html
to run this on your PC you will need to use DOSbox, a DOS emulator.
The controls in the original version are much more comfortable that those in the flash version.

Overview

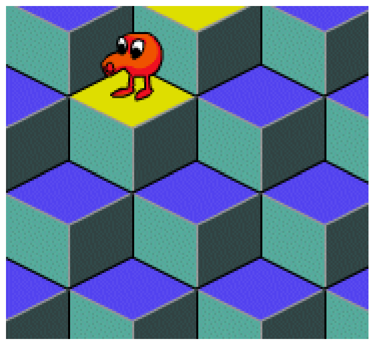
To make Q*bert a 3D game worthy of an OpenGL implementation several changes need to be made to the mechanics and the interface of the game.

The structure shape

instead of the single pyramid face of the original game, the 3d game structure will be a 3D diamond shape made of two full pyramids connected at their bases.


the general shape of the structure on which Q*bert is hopping

The structure is composed of little cubes, the same way the original structure is composed.



Like in the original game, each face of this diamond structure is made of cubes onto which Q*bert hops.

Free Rotation

The Original game's point of view is static and never changes. This is essentially what makes this game 2D, although the view itselt is isometric. In your fully 3D version you will need to enable free rotation to allow the player to view the structure from any angle. This feature is crucial for the user to be able to complete a stage. he needs to be able to look around the structure and search for cubes that are not in colored in the desired color.

Hopping around corners

In the original game, when Q*bert hops beyond the edge of the pyramid face he falls off into space. In out 3d version, the structure is not just a single face but a full 3D structure. Hopping beyond a corner or an edge would make Q*bert arrive at a different face of the structure. When this happens, the game should make sure that the point of view of the user also moves around the structure so that Q*bert can seen (View should rotate smoothly or in some other nice effect).

Notice that you should pay special attention to the exact mechanics of the hop beyond an edge. Think of the possibly outcomes of such a hop and decide where Q*bert need to end up standing when he arrives on the other side.

2D sprites VS 3D models.

In the original 2D game, all of the creatures on the screen, including Q*bert are simple 2D sprites. In the 3D game, they need to be actual 3D models which can be viewed from any direction. We will supply you with simple models in the form of OFF or OBJ files. You're welcome to use them or find other models more appropriate in your opinion.

Coloring cubes

In the original game, when Q*bert hops onto a cube he hasn't visited yet, the cube's top changes color, for instance from blue to yellow (these colors are different in every level). In the 3D game it would probably be better if the entire cube would change color and not just one face of it so that it could be seen clearly from every angle.

Enemies and elevators

Due to the fact that most of the basics mechanics of the game are changed, some changes need to happen to the way enemies of Q*bert move and the way elevators behave.

Minimal Requirements

- Implemented using OpenGL
- 3D Diamond shaped structure with 8 triangular faces.
- Dragging the mouse over the screen rotates the point of view
- Control of Q*bert is performed using the keyboard. make the controls as usable, playable and intuitive as possible.
- The Q*bert character and the enemies need to be 3D models which you load from a file
- The game needs to obey the general spirit of the Q*bert game rules. for instance:
    - hopping on cubes and coloring them.
    - Q*bert dies if he meets an enemy
    - a level is completed when all cubes are colored
    - etc'
- There should be atleast two types of model enemy and in addition, a ball enemy.
- At least one of the enemies needs to have some basic form of intelligence. he needs to be able to chase Q*bert around.
- The hopping of the characters between cubes needs to be animated. They shouldn't just disappear from one cube and appear on the next.

    - basic animation can be only gradual translation of the object between several points.
- The game needs to keep and display a numerical score for the player the same way the original game does.
- The game needs to keep track and display how many lives are left for the player.
    - once a certain score is reached a new life is added to the player.
- The game should have levels. once a level is completed, the player receives a new level.
- Different levels should have different cubes colors.
- Each new level should raise the difficulty of the game. Difficulty could be measured by:
    - the speed and number of enemies
    - the number of cubes on every face of the structure
    - etc'
- The appearance of gravity towards the center of the diamond. Q-bert should always be standing on its feet no matter where he is.

Bonuses

There are virtually infinite possibilities for extensions you can implement for this game. Any additional feature beyond the minimal requirements might grant you a bonus according to the degree of its complexity, the originality of the idea and quality of implementation.
Some ideas to get you started:

- Define a way to interact with elevators like in the original game.
- Define a way to be able to defeat the enemies in some way
- Construct additional 3D structures in addition to the 3D diamond structure.
- texturing on the structure and the characters.
- basic sound effects

Additional Notes

- Above all else, your game needs to be **playable** and **fun**.
- Alot of the functionality of this exercise remains undefined. You should show your creativity and define everything however you think is best.
- Game balance is a delicate art and it is sometimes missed even in commercial games.
    - The game should be difficult enough to be chalanging but not too difficult to fustrate the player.
    - It should be playable by a novice which plays it for the first time but also for someone who spent the last 5 hours playing.
    - The game should supply enough sensory input to not bore the player in grinding sessions
- As always, it is best to spend some time thinking about how you're going to solve some of the problems before you start writing a single line of code.
- Throught your coding, make sure you follow the Model-Display-Controller design pattern.

The following is a suggestion for implementation milestones you might follow:

- display a few simple polygons using OpenGL
- be able to rotate the scene using the mouse. this will be an invaluable tool for the rest of your work.

- construct the 3D diamond structure according to several parameters (color, size, etc')
- build a basic "Model" for the game as in Model-Display-Controller
- be able to display a small ball on top of cubes across the structure. The ball is a place holder for a character in the game
- implement the basic rules of the game - hopping from one cube to another
- be able to load a model from a file and display it in 3D (without the structure)
- replace the characters in the game with loaded models
- implement animation
- implement the rest of the game logic - enemy meeting, levels, lives, scores
- give the enemies intelligence (even a really silly one)

Submission

As before submission is in Pairs.
In the due date you need to submit the following:

- Full source code, packaged in a zip file
- Executables or JAR file which works when it is being double clicked.
- A short document (2-4 pages) explaining:
    - Instructions for the game
    - Features you implemented
    - The structure and design of your code
    - Anything else needed to make it work. Please supply

Please make sure you read the FAQ and Updates page frequently and before your submission.

Resources

How to install JOGL

Windows:

1. download JOGL from:
   http://download.java.net/media/jogl/builds/archive/jsr-231-1.1.1/jogl-1.1.1-windows-i586.zip
   Don't be tempted to download the AMD64 version. It will cause you trouble.
2. Save the zip file is your directory of choice, for instance in "**C:\Program Files\java**"
3. Extract the zip file into a directory - "**C:\Program Files\java\jogl-1.1.1-windows-i586**"
4. Create a new project in eclipse and add the startup code supplied below - Main.java.
5. Right click the project, go to properties->Java Build Path
6. Click "Add External JARs" and select the two JARs under "**C:\Program Files\java\jogl-1.1.1-windows-i586\lib**"
7. Do the same to add SWT.jar from wherever you installed it.
   After this the startup code should be able to compile but when you try to run it, it will likely fail with an exception. the JOGL DLLs are still missing.
8. Locate the directory of the JRE or JDK you are using. for instance "**C:\Program Files\java\jre1.6.0_02**". you can do that using the properties->Java Build Path dialog. look under the JRE to see where it is installed.
9. Copy the JOGL DLLs from "**C:\Program Files\java\jogl-1.1.1-windows-i586\lib**" to the \bin directory under of the JRE, in this example:"**C:\Program**

Files\java\jre1.6.0_02\bin"
This is the safest way to make java find these DLLs. there are other methods as well which invlove changing the PATH environment variable. do this at your own risk.

In the end, the DLLs and The JARs in your project should be of the same version. If you have other versions of JOGL installed for some reason, make sure that java uses the right ones.

After completing this the startup code should be working and displaying an OpenGL view.

Linux:
TO BE ADDED

Startup code in Java

This file is an initial framework for OpenGL.
It contains the basic entry points discussed in class and a simple scene made of two polygons for you to be able to see that OpenGL works.
feel free to change these files anyway you like.
Main.java

Models Loading

In this exericise you will work with OBJ model files.
You are required to load and parse OBJ model files and be able to display the model in them correctly.
The OBJ file specification can be found in these links:
http://www.martinreddy.net/gfx/3d/OBJ.spec
http://people.scs.fsu.edu/~burkardt/txt/obj_format.txt
http://people.scs.fsu.edu/~burkardt/data/mtl/mtl.html

Notice that you are in no way required to implement the entire OBJ format. The format has many features that you can safely ignore and your model will still render ok. The lines you probably must parse are the vertex definition (v), polygon (f) definition and material definitions.

Models

There are various sources from which you can aquire models.
You can either use one of the models supplied here, or you can download models from the internet or you can even create your own models.

This file contains a few models you can start with

Google sketchup is simple to use 3D modeling software. it allows you to design your own models from scratch. The basic version of google sketchup is free.
The "PRO" version of google sketchup also allows you to save the models you created into OBJ files. This version however is has a trial period of 8 hours.
http://sketchup.google.com/download/

Google also has a large collection of models created by sketchup users. some of the example models in the above file were created from models from this site:
http://sketchup.google.com/3dwarehouse/

There doesn't seem to be a way to have both the free and pro versions installed at the same time so I suggest that you start with the free version and when you want to convert your models to OBJ, upgrade to the PRO version.

Finally, there are other programs for instance such as "Deep Exploration" that are able to open and convert sketchup files into OBJ files.

links

http://www.eclipse.org/articles/Article-SWT-OpenGL/opengl.html
https://jogl.dev.java.net/

documentation:
http://download.java.net/media/jogl/builds/nightly/javadoc_public/
http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api/org/eclipse/swt/opengl/package-summary.html

A nice Tutorial to get you started
http://www.geofx.com/html/OpenGL_Eclipse/OpenGL_Eclipse.html
Notice that this tutorial talks about an *Eclipse plugin* and not a stand-alone applications so you will need to ignore some of the things mentioned there.