Exercise 3: OpenGL

In this exercise you will implement a 3D version of a game of your choice out of the three possibilities presented. The purpose of the exercise is not only to study the basics of OpenGL but also to get some experience in designing and implementing a real playable game.

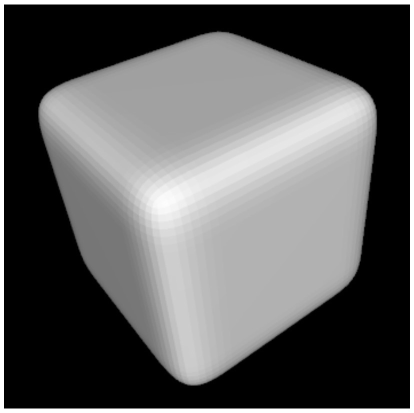
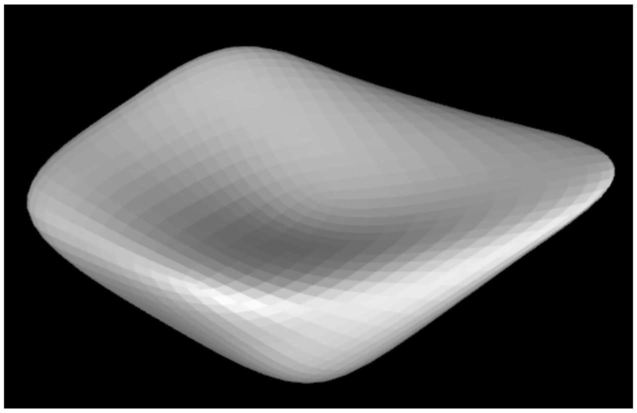
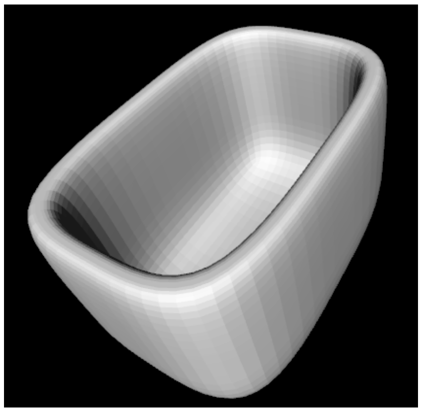
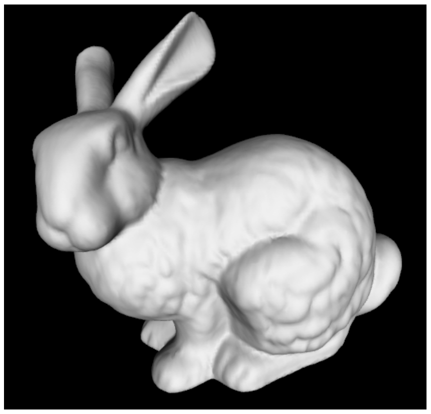
Overview

The three games presented below are classic PC games which are usually played on a 2D board. In this exercise you will implement one of these games but instead of a square 2D board, the game should be played on a general 3D surface mesh which we will call *"The world mesh".*

The basic requirements of your implementation are as follows:

- **World mesh** - (10 points) Loading and displaying, see details below.
    - Different levels of the game should use different world meshes.
- **Additional models** - (10 points) Loading and displaying the models that represent the player, enemies and any additional objects of the game.
- **Lighting** (5 points)
    - All surfaces displayed using Guorad shading
- **Camera control** (13 points) The game should automatically follow the player around the world mesh in a way that allows smooth and easy playing of the game
    - The view should be a perspective view
- **Animation** (20 points) all movements in the game, of the players and of the camera should be done in smooth animated transitions
- **Gameplay (Playability)** (40 points) This is an important focus of the exercise. Your game should be fun and easy to play and yet challenging and non-trivial
- High code quality (3 points)
- Original creative features (+5 point bonus)

A few examples of world meshes:

You need to choose ONE of of the following three games to implement. Before you start designing your game, play the 2D version for a while, get a feel of how the game should behave. The 2D versions are also likely to give you some ideas you can implement in the 3D version.
The game is played on the surface of the world mesh. Anything that usually happens in the 2D game somewhere on the game board can happen on the surface of the 3D world mesh.

When transferring these games to 3D there are a multitude of decisions to make as to how the game behaves and changes compared to the 2D version. You are given **full discretion** in this respect. You can go a bit wild with the game mechanics as long as it maintains a reasonable amount of resemblance to the original 2D game. Be creative!

The Games

Volfied/Qix/Xonix/ כובש שטחים

The premise of Qix is that the player's "space ship" need to capture and close most of the area of the surface it lives on. In 2D the player starts from the borders on a square and needs to capture atleast 85% or so of the square. In 3D the area that needs to be captured is the surface of the world mesh.

Minimal features this game should have:

- A way to move the player in the world using the keyboard or mouse.

- A way to capture and "paint" areas in a different color, preferably with some kind of "base" the player need to return to.
- Computer controlled enemies roam the world and harm the player if touched or if they touch the line the player makes. Some enemies can move randomly and some should have basic AI that lets them intentionally chase the player.
- Bonues packages that appear in the world that give the player special powers/points if captured.
- Finish a level if a certain percentage of the surface is captured.

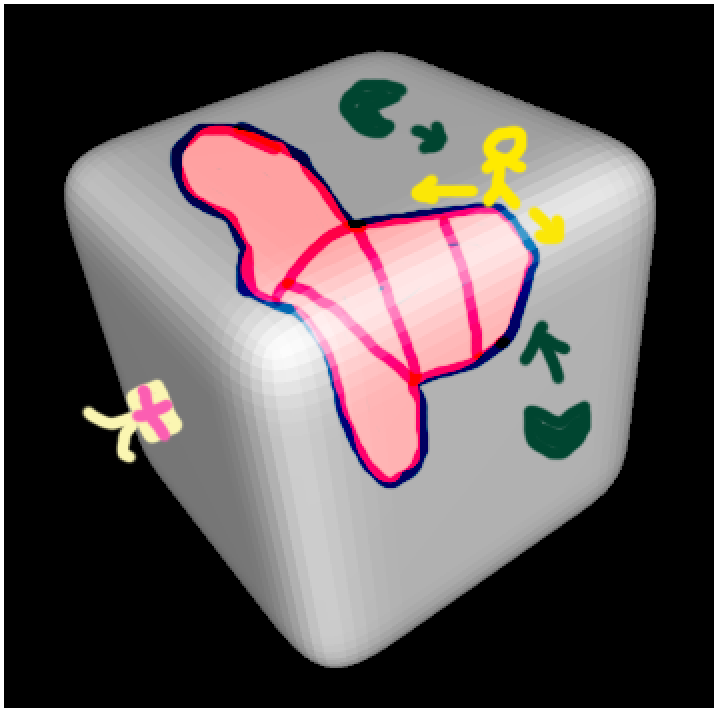
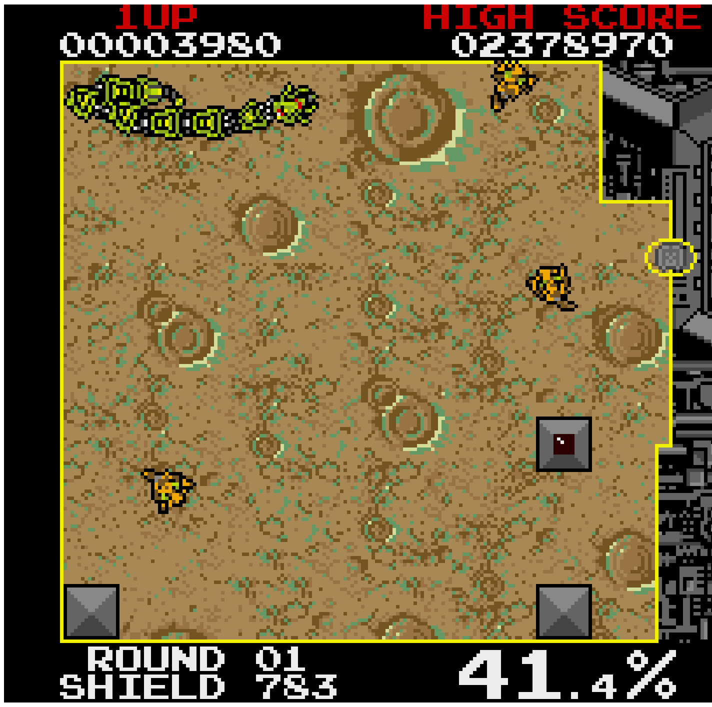
References:
Volfied: http://www.abandonia.com/en/games/811
To play this you will need the DOS emulator called DOSBox since it doesn't run on modern windows:
DOSBox: http://www.dosbox.com/download.php?main=1

Wikipedia on Qix: http://en.wikipedia.org/wiki/Qix

Flash Qix: http://drunkmenworkhere.org/flash/qix2.swf

A cute flash variant with a Pacman theme:
http://www.topoyun.net/oyun/12112/Flash-Volfied.html

A video of someone playing and finishing Volfied:
http://www.youtube.com/watch?v=GF5JcZ3zKdY

A silly drawing of what this can look like	A screen capture of Volfied, the 2D game.
	 1UP 00003980 HIGH SCORE 02378970 ROUND 01 SHIELD 783 41.4%

Light cycle

The light cycle came to be famous in the 1982 movie TRON. The basic premise is that there are a few players in the world, each riding a motorcycle/spaceship which is in constant motion forward, leaving behind them a trail of colored wall. If a player crashes into such a wall he loses the round. The winner of a round is the player that remains the last one. This is sort of like "snake" only that there are multiple players which aim to fail one another.

Minimal features:

- A way to guide the player in the world using the keyboard or mouse.
    - *Players should be able to move smoothly on top of the world mesh, not only on the edges!* See appendix 2 for more details on this.
- Draw the wall behind the player wherever it went. The wall may or may not disappear after a while, essentially making the player a sort of long snake.
- Computer controlled adversaries. These should have some basic AI that prevents them from dying immediately and possibly try to fail the player.
- Bonus packages that appear in the world. if a player takes the bonus he receives some special ability and/or points.
- Crashing into a wall terminates the player or adversary, the player wins the round if he remains last.

References:
Wikipedia on Light Cycle: http://en.wikipedia.org/wiki/Light_Cycle

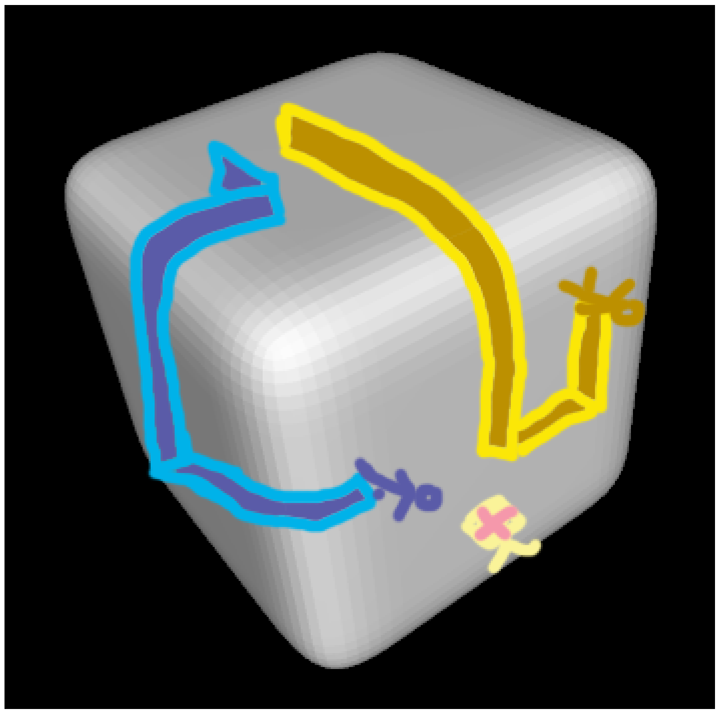
Nice flash version: http://www.fltron.com/index_flash.html

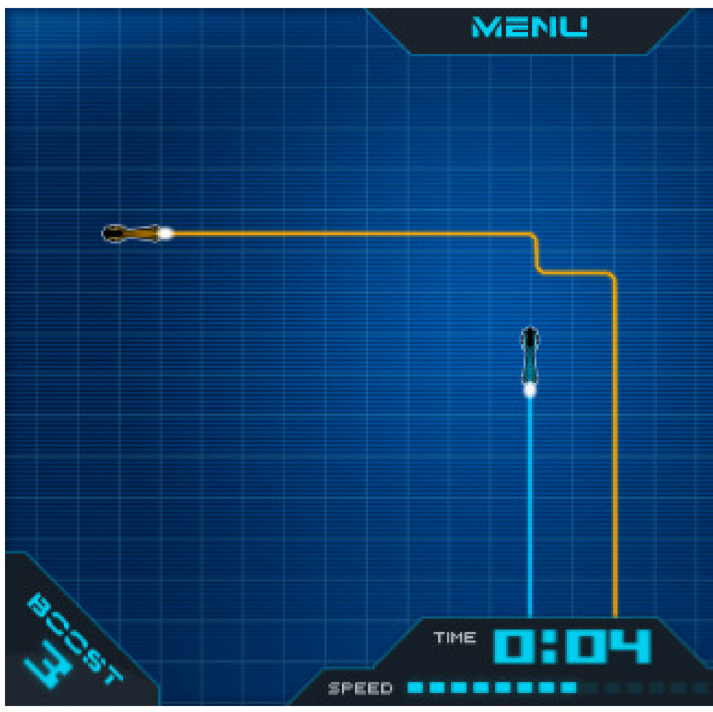
Zone0, A DOS version of the game: http://www.hotud.org/Action/Zona-0.html
(also requires DOSBox)

Original scene from the movie TRON (1982): http://www.youtube.com/watch?v=-3ODe9mqoDE
A scene from Tron 2, the PC game (2003): http://www.youtube.com/watch?v=geD9MNXmqs0

Another flash version (a bit silly): http://www.thepcmanwebsite.com/media/tron/

Another silly drawing of what this can look like	A screen capture of a 2D game version of the game





Scorched earth/Worms

This is basically a shooter with players scattered around the world, shooting each other until all but one are dead. In the 2D version of *Scorched earth* and *Worms* the view is a sideways view allowing the player to see the exact position of each one of the individuals playing the game. The focus of the game is on accurately hitting the area where the enemies stand rather than chasing them around the world.

Minimal features:

- A convenient User interface that allows the player to select a weapons, aim it and shoot.
- At least 3 kinds of different weapons.
    - At least one of these weapons fire some kind of projectile (a missle for instance)
    - Think of original weapons. the options are limitless.
- A simulation of gravity - projectiles that are shot upwards follow a ballistic path and hit the world surface at some point.
- A way to move the player. the player should be able to move slowly and not indefinitly. if a player moves he can't shoot.
- When a projectile hits the world it should have some effect on the world
    - For instance, a change of color, a change of texture, addition or removal of material etc' - there are some nice effects that can be achieved with very simple means.
- A number of adversaries with basic AI that aim to kill each other and the player.
- When a player is hit, he loose life points or die.
- Bonus crates/packages - If a player touches or hits a bonus object he gets some special ability, weapons and/or points.

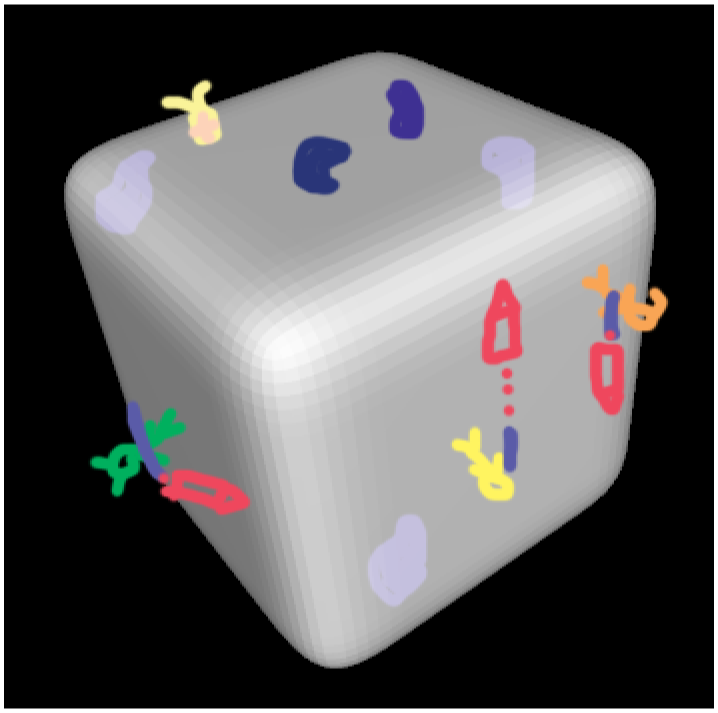
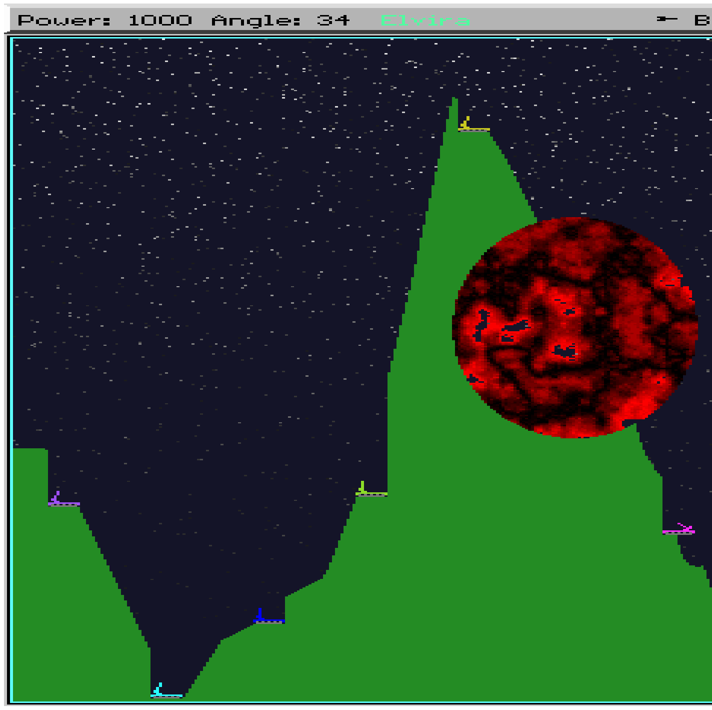
References:
Wikipedia on Scorched earth: http://en.wikipedia.org/wiki/Scorched_Earth_(computer_game)

Download Scorched earth: http://www.hotud.org/Action/Scorched-Earth.html
(also probably requires DOSBox)

Liero - a worms clone that takes a somewhat different approach: http://www.hotud.org/Action/Liero.html

A rather simple flash version: http://www.newgrounds.com/portal/view/255611
An online Java version: http://www.scorch2000.com/
A strange online multiplayer version: http://zwok-game.com/

A video of someone playing Scorched earth: http://www.youtube.com/watch?v=Io3LjftP6YQ&feature=related
A video of someone playing worms: http://www.youtube.com/watch?v=kI4QyvpJla8

An attempt to visualize what this could look like	A screen capture of the original 2D scorched earth
	

General Notes

Mesh Processing

A mesh is a collection of triangles or rectangles connected at vertices in 3D. A simple mesh for example is a cube with 6 faces and 8 vertices.
A standard data structure that can be use to hold a mesh is an Indexed Face Set (IFS). An IFS holds two lists:

- Vertices list - a list of 3D coordinates
- Faces list - A list of faces - triangles or rectangles - each face composed of a list of indexes to the vertices list.

An Indexed Face Set for example:

```
V = {  ( 1   1 0),
      (-1   1 0),
       (-1 -1 0),
       ( 1 -1 0),
       ( 0   0 1) }

F = {  [1 2 3 4],
      [1 2 5],
      [2 3 5],
      [3 4 5],
      [4 1 5] }
```

This IFS constructs a pyramid with one square face and 4 triangles.
The OBJ File format which you will work with contains a mesh in this form, first there is a list of vertices and then a list of faces, referring to the vertices by index. notice that the indexes start from 1.
The order of the indexes in a face is important. it determines if the face is front facing or back facing. the direction of the face [1 2 5] will be opposite than the direction of face [5 2 1].
Once you load the mesh into the data structure there are various things you can do with it which you will need to use in the exercise.

- Display it using OpenGL - simply go over the list of faces and create corresponding primitives using OpenGL. To avoid repeated costly calls to glBegin() and glEnd() you may want to save triangles and rectangles in seperate lists or draw every rectangle as two triangles.
- Calculate the normal of a Face - Needed for lighting. For a face [a b c] - taking the cross product (b-a)x(c-a). Since the order of the vertices is consistant, the direction of the cross product also remains consistant.
- Find All the faces that are adjucent to a certain Vertex. This is called the 1-ring of a vertex.
- Calculate the normal of a Vertex - Needed for gourad shading. Calculate the average normal of all the faces in the 1-ring of a vertex.
- Find All the faces that are adjacent to a Face. This is called the 1-ring of a face.
- Create and maintain a list of Edges or Half-Edges.

    - An Edge is an unordered set of two vertices that share two faces. In the pyramid above [1 2] is an edge. [2 1] is the same exact edge (unordered set)
    - A Half-Edge is an ordered tuple of two vertices that share two faces. [1 2] is a half-edge and [2 1] is a different half-edge.
- If you find it necessary you can implement a full Half-Edge structure. Read about it here:
    - http://www.flipcode.com/archives/The_Half-Edge_Data_Structure.shtml
    - http://www.cgafaq.info/wiki/Half_edge_implementation
    - While useful, this is not strictly required in the exercise and is left for your discretion.

The main use you will have for the mesh data structure is for the world mesh. The smaller models used for the players and enemies, while they are also small meshes on their own, do not require the full functionality described above so make sure you perform the heavy processing only for the world mesh.

Loading Models

Most of the model files available are OBJ model files.
You need to load and parse OBJ format to be able to display and do things with these models
OBJ is a very simple file format. the full 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
Material file format:
http://people.scs.fsu.edu/~burkardt/data/mtl/mtl.html
The .OBJ format is designed so that if you don't support a feature, you don't need to understand the lines that describe it. For instance, if you don't need to materials from the file, ignore the `usemtl'. If you prefer to always calculate the normals yourself, ignore the lines `vn' lines which specify normals. You don't need to understand every single line in the file.

Some of the models are in a differnt file format with extension .OFF
This file format is even simpler than .OBJ. more info here:
http://shape.cs.princeton.edu/benchmark/documentation/off_format.html

Meshes and Models

There are alot of various sources you can get models and meshes from for the world mesh and for the smaller players models. Following are a few useful resources.

This zip file contains a few models I collected or created: First collection Second Larger (different) collection
A collection of alot of models, usually used in academic geometric modeling papers: http://shapes.aimatshape.net/viewmodels.php?page=1

Princeton shape benchmark, quite a few models, most are simple, some are rather silly. http://shape.cs.princeton.edu/benchmark/
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.
KawaiiGL This is a small program that allows you easy modeling using a simply language syntax. the meshes at the top of this page were created using this program in just a few minutes. Get it from **this link** • To activate it you probably need to install the file "vcredist_x86_2008sp1.exe" from the zip file (this is the Visual Studio 2008 SP1 Runtime libraries) • Right click the text are to get some ready made models • play with the check-boxes to tweak the view • When points are visible - right click and drag a point to deform the shape. • Click "Save" to save as an .OBJ file.   ◦ To save a deformed shape, go to the "Current" Tab in the edit window, copy the text, paste it in the edit tab and then save. Yes, this is a bug. • Pressing ALT will show you occluded points.

Important Note:

Pay close attention to the resolution of the models and meshes.

A player model that is going to be small on the screen doesn't need to have thousands of faces. this will just make your game slow and memory hungry.

Animation

A call to canvas.repaint() causes SWT to schedule a paint event to the window.
A simple way to manage the animation of a scene is to set up a single timer the fires every few milliseconds, updates the data structures maintained by the game and makes a call to repaint(). In the display() override the game should draw the current state according to the data structures maintained.
A simple example for this is a simple ball that moves on the screen.
The object starts from Point A and after 2 seconds need to end up in point B. Let t be a variable that control the position of the ball. When t==0, the object is displayed at A, when t==1, the object is at B. For any value between 0 and 1 the object will be at the corresponding point on the line between A and B.
To set up this animation we create a time that fires every 25 milliseconds, 40 times a

second. We initialize t=0 and Every time the timer fires we increment t by 0.0125 and call repaint(). This way after 80 times the timer fires - after 2 seconds - t will reach 1.0 and the object will be displayed at B.
In this way we can manager multiple animations with just a single timer. Don't create multiple timers for every animation since every timer involves an overhead which accumulates rapidly.

Moving in the world

Most of the models in the scene are usually drawn on the surface of the world mesh. This includes the player model, enemies models and bonus objects. The projectiles of scorched earth however are an example of models that fly above the surface of the world mesh.
When a models is situated on the mesh its orientation need to match to that of the face (or faces) it is standing on. This can be achieved by rotating the model to the orientation of the normal of the face, as I explained in class.

Camera Control

As the player model moves around the world mesh the camera needs to follow him wherever he goes so that the player always know what's going on. This means that the model should rotate and possibly translate automatically so that the player model is always visible.
For instance if the world mesh is a cube, when the player moves to the back face of cube, the cube should rotate to make the player visible.

Gameplay

Your game should be fun to play. Although this is a subjective measure there are a few traits a good game usually have.

- Usability - the user interface is natural and accessible. A user shouldn't have to read the manual to be able to play the game. The user interface should be simple and as much as possible self explanatory.
- Difficulty - The difficulty of the game should be proportional to the level of the human player. An overly difficult game will frustrate the player into closing it with rage while a game that is too easy will bore him into doing something else. The simplest way to implement this concept is by making advanced levels of the game harder and harder to complete.

Implementation

We recommend you work with Java and JOGL for OpenGL support. You are welcome to implement the game in C++/C#, but we can't guarantee support.

Startup code

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 so you could see that OpenGL works.
feel free to change theis file anyway you like.
Main.java

Submission

As before submission is in Pairs.
In the due date you need to submit a **single** zip file with the following:

- Full source code
- JAR file.
- 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 submit your work to **shoosh.cg@gmail.com**

Send One email containing everything you submit and wait for a confirmation. if you don't receive a confirmation response after two days send again.

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

More useful 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.

Appendix 1: Installing JOGL

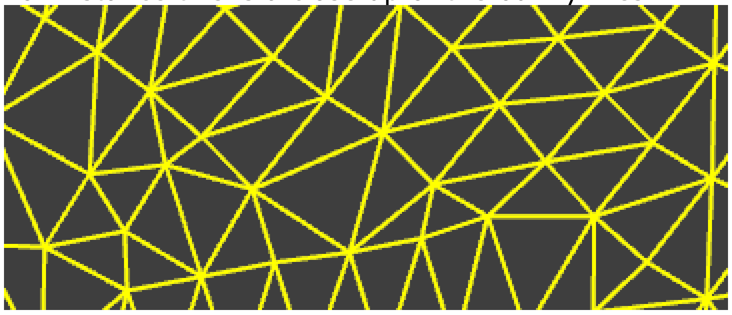
Windows:

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 nothing but 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.

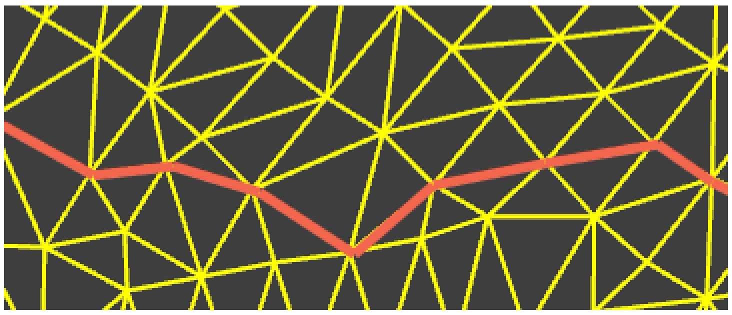
In the end, the DLLs and The JARs in your project should be of the same version. If you have other versions of JOGL installed for some reason, make sure that java uses the right ones.
After completing this the startup code should be working and displaying an OpenGL view.

Appendix 2: moving smoothly on a mesh

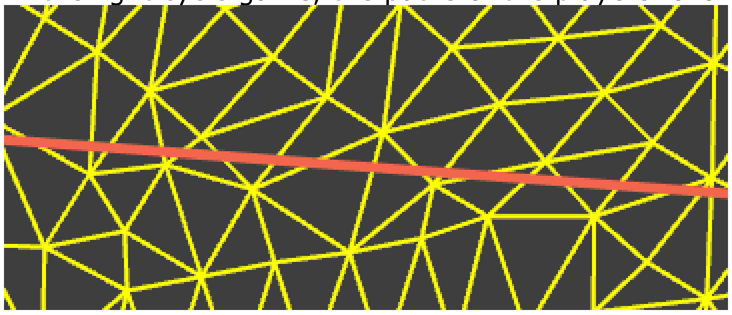
The faces of a general mesh may not be aligned in such a way that allows moving in a straight line over it.
For instance this is a close up of the bunny mesh:



If you move the player model only on the edges of the mesh, its path will not be straight:



In the light cycle game, the paths of the players follow need to be straight, like so:



The way to implement this is remember that the "player" is at all times, either on the edge between two triangles, on a vertex between a number of triangles, or on the surface of one. Lets assume that the player is on the edge between two triangles and has a movement vector towards one of them. Its easy to project the movement vector onto the plane which holds the new triangle and get the new movement vector for the player. If you do an intersection calculation between this vector and the triangle, you'll get the entry/exit points and know when you need to recalculate his movement vector. Of course if the player presses a key and changes the movement direction mid-triangle, you need to change the movement vector and calculate a new exit point.