Computer Science

# COMPSCI 372 S2 C - Assignment 2

Due Date: Monday 22nd September 2008, 9.30pm

**This assignment is worth 6.25% of your final grade.**

In this assignment you will create a little game containing a plane with grass, 20 flowers, Easter eggs falling from the sky, a robot with a basket at the end of its arm and a reflecting mirror. The objective of the game is to catch the Easter eggs with the basket. If you hit an egg with the rim of the basket or if you miss it the egg gets smashed [please feel free to extend the game and add appropriate sound effects ;-) ]. The game finishes after 50 eggs have been smashed.

Download the file Ass2.zip which contains a .NET solution including all necessary source files. Unzip the files and compile and run them. You should see a grass covered plane, 20 flowers, a non-reflecting mirror, and Easter eggs falling from the sky.

NOTE: In order to make debugging easier and improve the understanding of texture maps, all texture images are stored in Portable Pixel Map format (ppm). This format is very inefficient to load and your program might take a long time to load when run in DEBUG mode. You can avoid this by compiling your program in RELEASE mode.

## *1. Peerwise (8 Marks)*

The program Peerwise enables students to submit and answer multiple choice questions and to rate and comment on them. The idea is to encourage students to not only test their knowledge by answering questions, but also to gain a deeper understanding by devising new questions and plausible, but incorrect, multiple choice answers.

**(a)** **[2 marks]** Log onto the Peerwise page (http://peerwise.cs.auckland.ac.nz/UoA/) for COMPSCI 372 and **answer two questions** and rate them.

**(b)** **[2 marks]** Log onto the Peerwise page (http://peerwise.cs.auckland.ac.nz/UoA/) for COMPSCI 372 and **submit one new question** (it must have a correct answer, 3-4 wrong choices and an explanation for the correct answer).

**(c)** **[4 marks]** Please answer the following questions and write your answers (and additional comments) into a Word or pdf-document titled `Peerwise.doc` (or `Peerwise.pdf`):

(i) Do you think that answering other students' questions improved your understanding of the topics taught in this course? Please explain why or why not.

(ii) Do you think that formulating new questions for Peerwise improved your understanding of the topics taught in this course? Please explain why or why not.

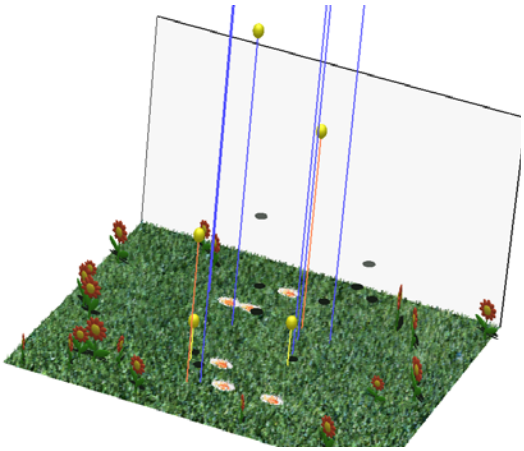
(iii) How did you decide what question to submit for this assignment? Did the interface (i.e. text input is easier than graphics or formula input) influence your choice? How much time did you spend thinking about an interesting question?

(iv) Do you think Peerwise is as suitable for a Computer Graphics course as for stage 1 papers? Please explain your answer

(iv) Please submit any other comments you have to our use of Peerwise ☺

## 2. 3D Geometry & Transformations (8 Marks)

**(a) [2 marks]** When playing the finished game you will find that it is hard to catch eggs since it is difficult to perceive the exact 3D position of an egg and where it will land. Implement the method `drawPath()` in the `CEgg` class, which draws a line from the centre of each egg to the ground. A line is coloured blue (0.3, 0.3, 1.0) if the distance to the ground is larger than 3 units, orange (1.0, 0.5, 0.3) if the distance is between 1.5 and 3 units and yellow (1.0, 1.0, 0.3) otherwise. Visibility of the lines is toggled by pressing the 'h' key (already implemented). After completing this exercise your programs should look similar to the screen shot on the right:

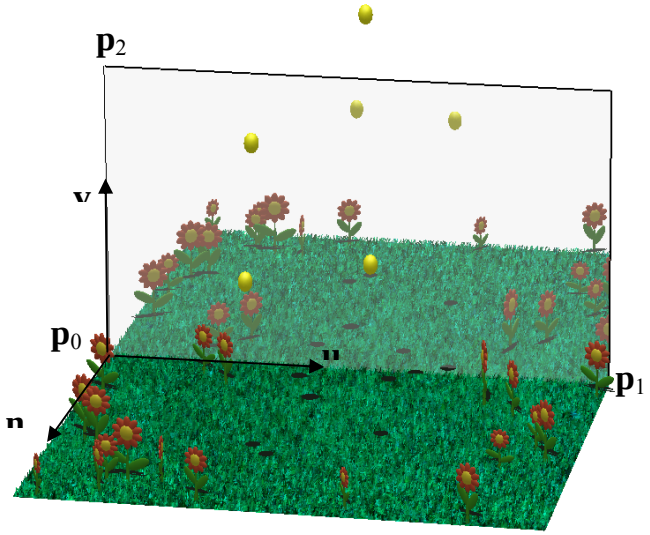


**(b) [6 marks]** In this question you will complete the code for implementing a mirror. The complete implementation is a bit tricky: the reflected scene is drawn such that it is only visible in the polygon representing the mirror (this is achieved using a *stencil buffer*). Furthermore the light sources are mirrored and the reflected image is blended with the grayish colour of the mirror.

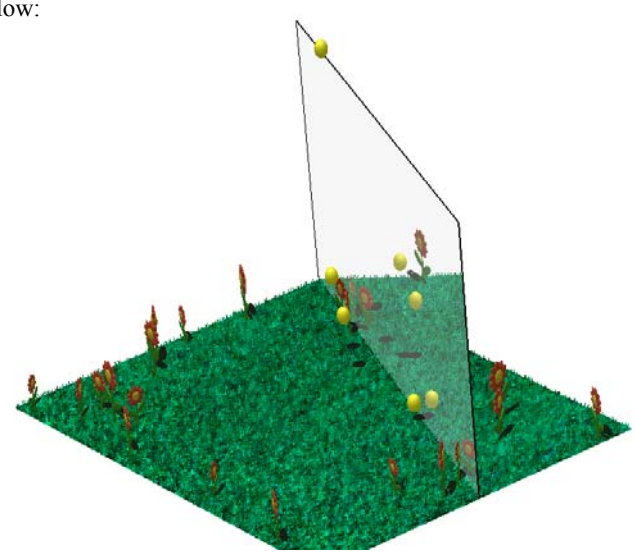
Your task is to implement the functions `initReflectionMatrix()` and `initWorldToMirrorTransformation()` in the file `CMirror.cpp`. The first function initialises a reflection matrix, which reflects an object on the *xy*-plane, i.e. it inverts the *z*-coordinate of any point. The second function initialises a matrix, which transforms an object from the *uvn*-coordinate system of the mirror, defined by the points $\mathbf{p}_0$, $\mathbf{p}_1$ and $\mathbf{p}_2$, to the xyz-coordinate system in which the scene is defined. In order to do this the matrix must perform a translation by $\mathbf{p}_0$ and a rotation of the *uvn*-coordinate system into the *xyz*-coordinate system (Hint: Such a rotation is described in your lecture notes!)

The *uvn*-coordinate system of the mirror is illustrated below:

0 hits
6 misses

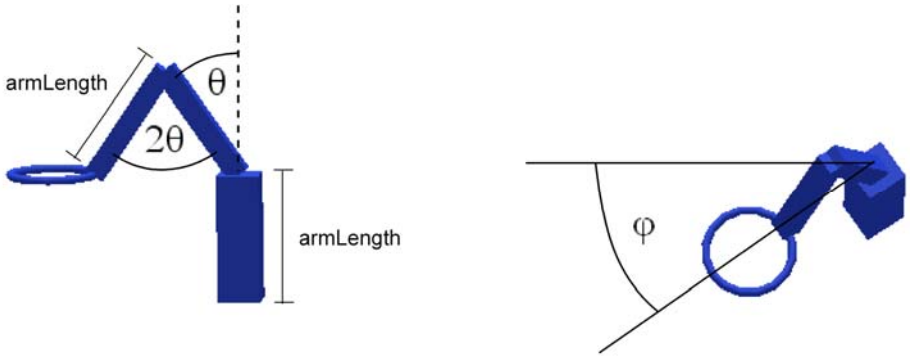


**Note:** Since the mirror is parallel to the xy-plane it is very easy to specify the required matrices directly. In order to get full marks you have to find the general transformation matrix for any values of **p0**, **p1** and **p2** of the mirror. For example, if you change $\mathbf{p}_1$ to (2.5, 0, 0) you should get the result shown below:



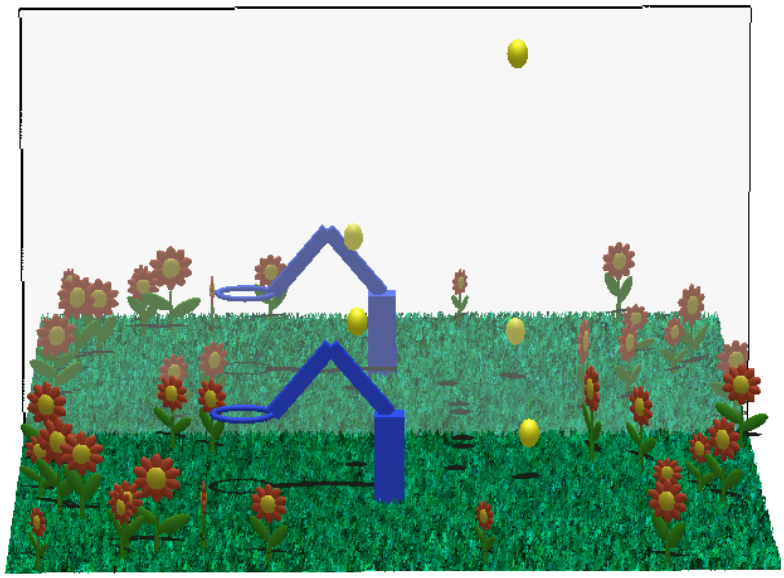
## *3. Modelling with Polygon Meshes (12 Marks)*

**(c)** [6 Marks] Define the robot shown in the image below. The robot has three arms with lengths `armLength` and at the tip of the 3rd arm it has a torus which is always parallel to the ground plane. The outer radius of the torus is `basketRadius` and its inner radius is `rimRadius`. The bottom arm is perpendicular to the ground plane and centered at the origin. The whole robot is rotated by an angle φ (phi) around its centre axis. The second arm is rotated by the angle θ (theta) with respect to the bottom arm and the third arm is rotated by 2θ with respect to the second arm as illustrated in the figure below.



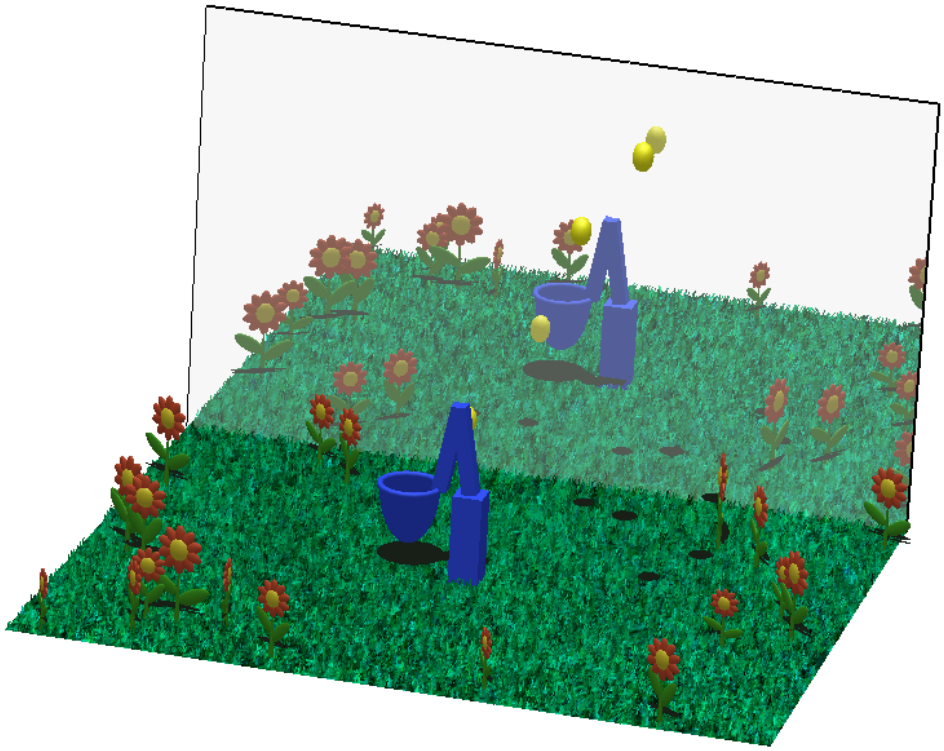

At the end of this exercise your robot should look like in the picture below. Note that the keys 'a','s','w' and 'z' are used to modify the angles φ and θ, i.e. at the end of this exercise you should be able to move the robot using these keys.

NOTE: For this exercise it is sufficient to use matrix operations (translation, rotation, scaling, push, pop) and two solid objects defined in GLUT. In order to make it easier for you to test your implementation I have defined a variable `basketCentre` which gives you the point at the centre of the torus. You can check your code by drawing this point and by verifying that it is indeed in the centre of the basket rim of the robot.



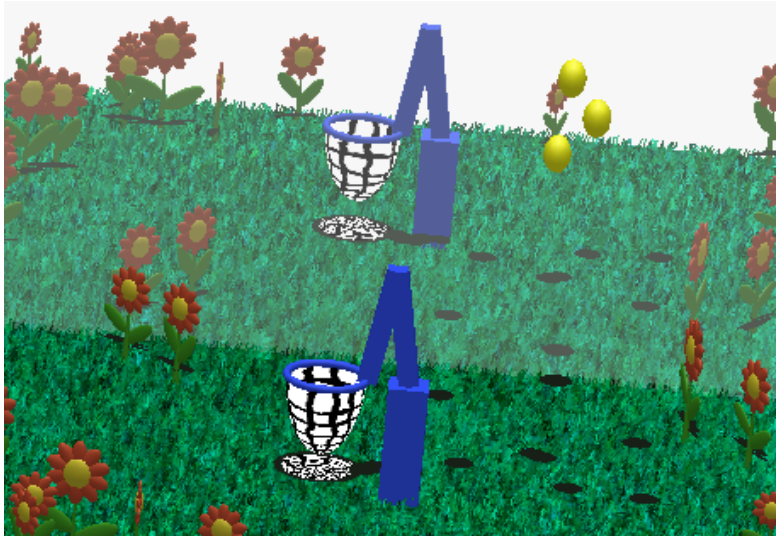
**(d)** [6 Marks] Complete the function `_drawBasket()` and add it to the draw method of the `CRobot` class so that it draws a basket defined as a **surface of revolution**. Please precompute the vertices of the basket's surface in the constructor of the `CRobot` class and draw the surface using quad strips as indicated in the figure below. Since we later on use texture mapping for the basket it's **not necessary** to define surface normals.
You must define the surface of revolution yourself – using GLUT commands or other geometry libraries gives automatically zero marks. The dimensions and number of vertices of the surface are defined in the code by constants explained in the illustration below.


At the end of this step your scene should look similar to that in the image below.



## *4. Texture Mapping* *(8 Marks)*

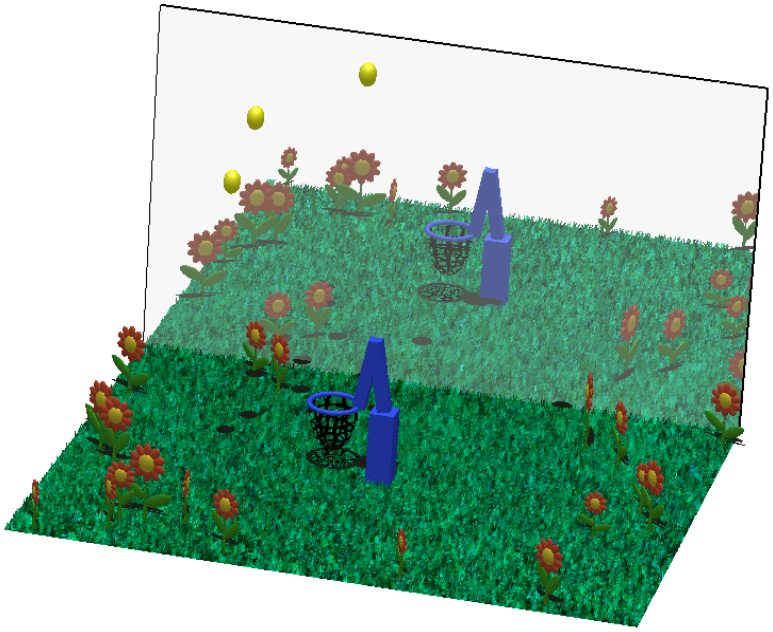
**(a)** [5 marks] Use texture mapping to draw the basket at the end of the robot arm. Complete the missing code so that it loads the texture map "Net.pgm" and maps it onto the basket. Note that the program contains already a routine to load a texture of type 'Portable Pixel Map' (ppm) - a 'Portable Gray Map" (pgm) is defined similarly but has only one byte per pixel (a greyscale value in the range 0 to 255).
Note that the texture image has been constructed such that the pixels on its left and ride side are identical. Consequently you should map the horizontal direction of the image onto the circumferential direction of the basket's surface in order to get a continuous texture.
At the end of this step your scene should look something like this:



**(b)** [3 marks] Modify your code such that white pixels in the texture image become transparent. In order to do this you have to set the fourth coordinate (the "alpha value") of all white pixels to 0. You also have to enable blending and alpha testing which is done by inserting the lines

```
glEnable(GL_BLEND)
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glAlphaFunc(GL_GREATER, 0.9f);
glEnable(GL_ALPHA_TEST);
```

before performing the texture mapping. You don't need to understand blending and alpha testing, but if you are interested feel free to read more about it in the OpenGL programming guide. Don't forget to disable blending and alpha testing at the end of the draw method. After implementing this step your scene should look something like this:



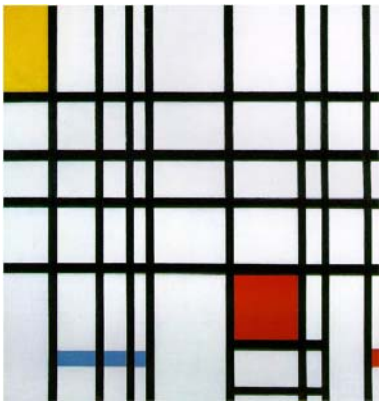
## *5. Bonus Question – Informative Art* *(6 Bonus Marks)*

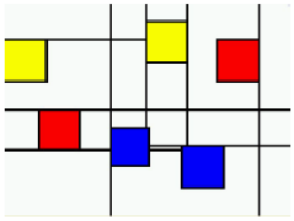
At a conference some years ago a Swedish group (Lars Erik Holmquist and Tobias Skog, Future Application Lab, Viktoria Institute, Göteburg, Sweden) presented a paper titled 'Informative Art: Information Visualization in Everyday Environments'. The idea behind this paper was to combine a dynamically updated information display with the decorative role of visual art, such as posters and paintings.
In this question you should develop an animation displaying 'Informative Art'. You are marked according to creativity (i.e. select a suitable piece or art to display the information and create an animation inspired by it), artistic skills (does your animation look like 'art'?) and technical skills (the quality of your animation).

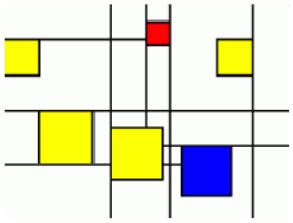
Here are two examples from the original paper:

**Example 1:** The image below on the left is a painting from Piet Mondrian (Composition with Red, Yellow, Blue - 1921). The authors create an interactive display similar to this painting consisting of 6 squares indicating the weather in 6 cities. The size of a square indicates temperature and its

colour indicates sunshine, cloudy weather or rain. The two images on the right show two instances of the resulting interactive display.







**Example 2:** The image below on the left is a work from the American 'Pop" artist Andy Warhole (100 Campell's Soup Cans). The authors create an interactive display similar to this work representing a count-down clock or "egg-timer". The interactive display uses asparagus soup (yellow) and tomato soup (red) cans. When the countdown starts all cans are asparagus soup cans and at each time step one asparagus soup can is replaced with a tomato soup can. The images on the right show two instances of the resulting animation.

**Your task:** Create an "informative art" animation. Do **NOT** copy the above ideas or ideas developed by anybody else. Look for a suitable data set you want to visualize and find a suitable art piece you want to use for this visualization. Submit a Word document `InformativeArt.doc` which explains the data set you visualize, the art piece you chose and how the art piece is modified to represent this information. Submit a program with the main method `InformativeArt.cpp` which implements your idea. If you submit several files please create a zip-file `InformativeArt.zip`. **Please note that this bonus questions will only be marked if you provide a working implementation!**

☺ *Enjoy!*

## *Assignment submission*

**All files must include your name, UPI and ID in a comment at the beginning of the file. All your files must be able to be compiled in the stage 3 lab without requiring any editing – any file which doesn't compile results in zero marks.** In particular any file with OpenGL commands should include the corresponding OpenGL libraries as:

```
#include <gl/gl.h>
#include <gl/glu.h>
#include <gl/glut.h>
```

All your files should include adequate documentation.

The assignment due date is Monday, the 22nd September 2008 (time: 9.30pm). Late assignments are not accepted.

Please submit:

- a document `Peerwise.doc` (or `Peerwise.pdf`) with your answers to question 1 (c).
- the file `Egg.cpp` with the missing code completed.
- the file `Mirror.cpp` with the missing code completed.
- the file `Robot.cpp` with the missing code completed.
- any other files you have changed (NOTE: The best solution does not require changes to any other files).
- if you do the bonus question the files `InformativeArt.doc` and `InformativeArt.cpp` or `InformativeArt.zip`.