# CSC 433 / 533: Spring 2017
# Assignment 2

**Assigned:** Thursday, Feb 14th
**Due:** 11:59pm Tuesday, Mar 7th (new due date)

For this assignment you will write a object viewer which is an interactive program that reads geometry descriptions from files and uses OpenGL to render the objects in 3D. Programs such as these are also called mesh viewers. Your program will be graded on the Mac workstations in the Gould-Simpson building.

For all programs this semester:

1. Provide a Makefile that builds the program. We will not grade programs that do not compile.
2. Your directory needs to contain all source code for both the application on the CPU and shader files for the GPU, and required data files.
3. Report any errors to stderr and provide a usage message when the program is invoked with no scene file name.
4. Include a readme.txt file and include any special instructions or assumptions.
5. Comply with the University of Arizona Code of Academic Integrity. Review the course policies at https://www2.cs.arizona.edu/classes/cs433/spring17/policies.html

## Tasks:

1. **viewer**
    a. In your makefile create an executable named ***viewer***.
    b. Input to the ***viewer***:
        a. The program takes one parameter which is the scene file name, for example

            viewer teapotScene.txt

        b. The format for the scene file is described later in this document.
    c. Read and display geometry described in the wavefront obj format. The geometry information is in *.obj files and the color information is in .mtl files. Some URLs for this information are (http://www.martinreddy.net/gfx/3d/OBJ.spec , http://www.fileformat.info/format/wavefrontobj/egff.htm , http://www.fileformat.info/format/material/ and http://paulbourke.net/dataformats/mtl/ .

d. I'm providing you with source code to read Wavefront obj files and provide descriptions that can be used to draw the objects
    i. A prototype for this function is in loadObj.h
    ii. loadObjFile has the following constraints:
        1. process only the ASCII format of an obj file and only process polygonal descriptions.
        2. In an obj file, the name of the material library, is a file name that is located in the same directory as the obj file.
        3. Note: the indices used to describe vertex connectivity in the obj file start with 1 not 0.
        4. It will ignore many pieces of information in the obj file. loadObj processes the following keywords of the obj file
            a. mtllib and usemtl describe the surface characteristics
            b. v, vt and vn for vertex information
            c. f for vertex connectivity that describes a triangle (or face)
        5. n-sided polygons. In this assignment, you will only deal with obj files that describe objects with **triangles**. You will find many obj files that have more than three vertices in a face description. For those that are interested, I have read that blender, a free 3D modeling program, offers a triangulate feature that could be used to read an obj file with n-sided polygons and create an obj file with only triangles. This is **not** a required part of the assignment.
        6. For an obj file that doesn't provide normals, face normals are computed by taking the cross product of two vectors along the sides of the triangle, and uses that normal for all three vertices of a face
    iii. To process a VAO created by loadObjFile using the provided Blinn-Phong shader the following layout statements are needed

```
layout(location = 0) in vec3 vertexPosition;
layout(location = 1) in vec3 vertexNormal;
```

    i. There are many .obj files on the web, here are some repositories:
        1. http://people.sc.fsu.edu/~jburkardt/data/obj/obj.html .
        2. https://www.turbosquid.com/
        3. tf3dm.com
    ii. Respect people's requests about using their geometry files.

2. Perform lighting calculations in eye coordinates rather than world coordinates. But note that the light coordinates provided in the scene file are world coordinates.
3. Support one shader
    i. Blinn-Phong
4. Specification and control of 3D viewing parameters.
    i. Height in world coordinates is the z axis, the x-y plane is the floor.
    ii. use a perspective view

5. keyboard commands
    i. ESC and q – quit application
    ii. p – reset view to default values
    iii. o – toggle between wireframe and solid rendering, default is solid.
    iv. Use the bounds of the scene to scale the distance for key presses.
    v. Dolly in/out or truck in/out
        1. w,s - w moves eye (camera) and focal point forward and s moves them backward. This is called dolly in/out or truck in/out. The motion is along the gaze vector. Move both eye and focal point by the same amounts. GLFW_KEY_UP does the same as w and GLFW_KEY_DOWN does the same as s;
        2. a,d move the camera and focal point left and right, or truck left/right. a moves the camera and focal point left, and d moves the camera and focal right. GLFW_KEY_LEFT does the same as a and GLFW_KEY_RIGHT does the same as d.
    vi. Pedestal up/down
        1. r,t – r moves the camera and focal point up and t moves the camera and focal point down
    vii. Rotate viewing
        1. z,x - rotates the eye position and focal point around the view up vector, z rotates 1 degree clockwise, and x rotates 1 degree counter-clockwise
    viii. view up vector movement
        1. c - rotate view up vector by 1 degree counter-clockwise around gaze vector
        2. v – rotate view up vector by 1 degree clockwise around gaze vector
6. Additional GLFW call
    i. Add GLFW callback, window size callback. From this callback you will get the new width and height. Use this information to compute an aspect ratio, width/height. Use this new aspect ratio to modify the perspective transformation, set by either a call to glm::perspective or glm::frustum.

**2. Overview of program**

1. By reading a scene description, multiple objects can be placed within a 3D environment (within a common world coordinate system). Modeling transformations position the objects, and camera specification determines the view. Up to 4 lights can be included in the scene.
2. To set up depth buffering (z buffering).
    i. Mandatory: enable depth testing with glEnable( GL_DEPTH_TEST).
    ii. Recommended: enable depth mask (or writing to the depth buffer) with glDepthMask (GL_TRUE).
3. Add a reshape method, set up with `glfwSetWindowSizeCallback`, to support resizing the output window.

**3. Hints**

a. You can read and display an OBJ file before you process the scene file. You can call loadObj and read a static file name with no modeling transforms. Once that is working then you can add reading the scene file and handling modeling transforms for objects.

b. loadObjFile will read an OBJ file and return a list of meshes in the OBJ file separated by material. Maintain a master list of objInfo records in your program of all the objects that have been read. Code could be something like this:

```
loadObjFile(objFileName, &theseObjects, &objCount);
// copy the object information into the master list
for ( int i=0; i < objCount; i++ )
   masterList.push_back(theseObjects[i]);
```

where masterList is an STL vector of objInfo records. You do not have to use an STL vector, but it is convenient.

c. Light positions are in world coordinates, but the shaders do light calculations in eye coordinates. You will need to convert light descriptions to eye coordinates

d. To compute the normal matrix, you can use the upper 3x3 of the MV (model view matrix). This means that we cannot do non-uniform scaling, and we don't support shear in the modeling transforms. If you want to do the general case, you can use the transpose of the inverse of the MV matrix, as glm supports these operations.

e. Remember that you will be transferring 4x4 and 3x3 matrices to the shader(s). Use the correct glUniformMatrix call for these transfers from CPU to GPU.

f. Transferring light information to the shaders is tedious, you might find it useful to use code like this

```
for (int l = 0; l < numLights; l++)      // for each light in the array
{
sprintf(uniformName, "Lights[%1d].isEnabled", l);
loc = glGetUniformLocation(shaderPrograms[BLINN_PHONG], uniformName);
glUniform1i(loc, Lights[l].isEnabled);   //turn this light on

sprintf(uniformName, "Lights[%1d].isLocal", l);
loc = glGetUniformLocation(shaderPrograms[BLINN_PHONG], uniformName);
 glUniform1i(loc, Lights[l].isLocal);
   ...
sprintf(uniformName, "Lights[%1d].ambient", l);
loc = glGetUniformLocation(shaderPrograms[BLINN_PHONG], uniformName);
glUniform3f(loc, Lights[l].ambient.r, Lights[l].ambient.g,
 Lights[l].ambient.b);
```

4. **Grading rubric – 100 points total**
    1. [30 points] Correctly process scene files, including view settings, up to 4 lights, and multiple obj files. You can assume that the scene files are formatted correctly, i.e., you do not need to include error processing for misspelled keywords, out of range values, etc.
    2. [25 points] reading and displaying obj objects.
        i. View one simple object. [5 points]
        ii. View one moderately complex object. [5 points]
        iii. Viewing multiple objects with individual modeling transforms. [5 points]
        iv. Reshape display window. [5 points]
    3. [25 points] control of viewing parameters, including camera, look at point, and view up direction
        i. Control of camera or eye position, focal point position, and view up vector. Correctly respond to all the keyboard commands listed in the document. [20 points]
        ii. Smooth and intuitive camera movement. [5 points]
    4. [10 points] Create an interesting scene with a minimum of 1) 4 objects and 2) 2 lights.
    5. [10 points] For the written assignment answering the questions below.

**Questions**

6. [3 points] Show the viewing matrix for the following settings:
   i. Eye = 5,0,0
   ii. Focal point = 0,0,0
   iii. View up = 0,1,0

7. [4 points] For the world coordinate 3,0,0, what is its value in eye coordinates using the viewing matrix in the problem above?

8. [3 points] Compute the face normal for a triangle described by these vertices (0,0,0), (1,0,0) and (0,1,0) in this order. Assume CCW ordering of these coordinates.

## Scene File Format

An overview of the format is a collection of sections that are specified with keywords and values. The sections are 1) view, 2) lights, and 3) objects. Allow for a maximum of 4 light sources.

View section: keyword **view**

  **eye** <vec3>
  **center** <vec3>
  **viewup** <vec3>

Lights section: keyword **light**

  **type** [local|spot|directional]
  **ambient** <vec3>
  **color** <vec3>
  **position** <vec3>
  **constAtt <**float>
  **linearAtt** <float>
  **quadAtt** <float>
  **coneDirection** <vec3>
  **spotCosCutoff** <float>
  **spotExponent** <float>

Objects section:

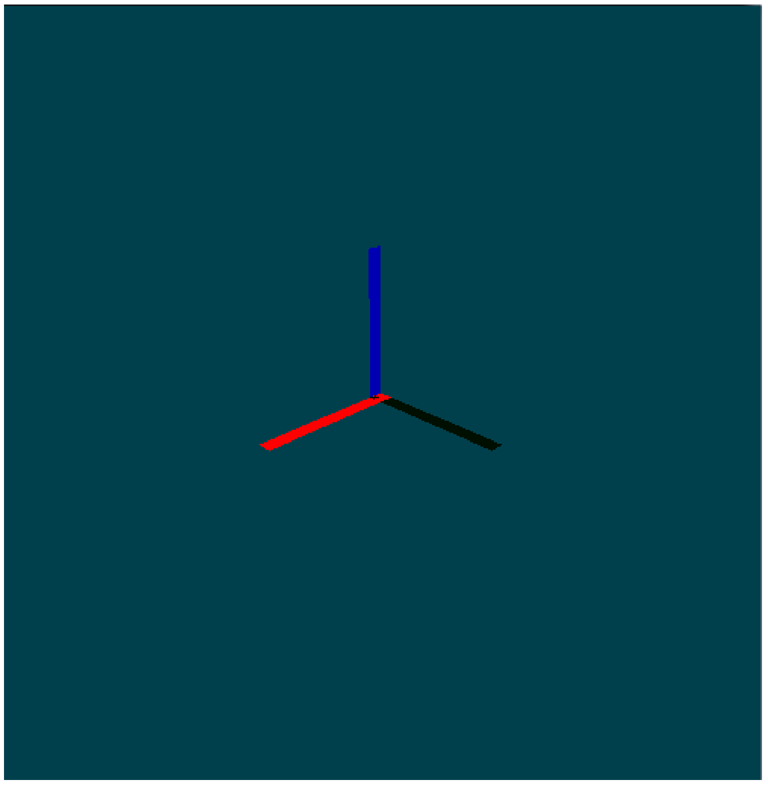
  **object** <filename of Wavefront OBJ file>
  **shader** [Blinn_Phong|COOK_TORRANCE|GOOCH]
  **[**
  **rx** <angle (degrees)> **ry** <angle (degrees)> **rz** <angle (degrees)>
  **t** <vec3>
  **s** <vec3>
  **]**

A light description can have a variable number of lines, depending on the type of light. Position is actually direction for a directional light source, and actual location for spot and local light sources.

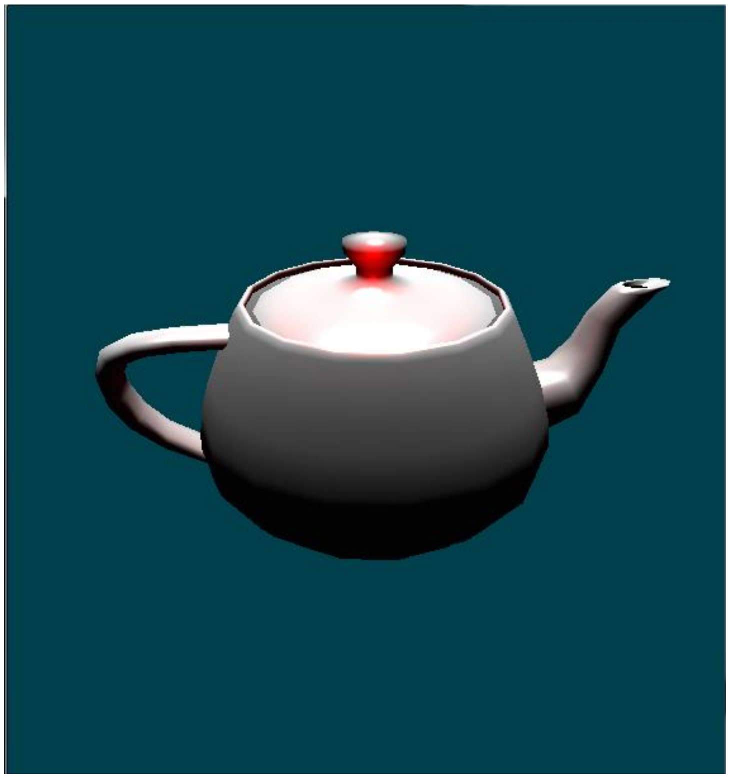
Modeling transforms for objects are not required, in which case the modeling transform for the object will be the identity matrix. The transformations are to be applied in the order listed in the file.  There is not limit of the number of canonical transforms that can be applied.

Use forward slashes for paths to OBJ file descriptions.

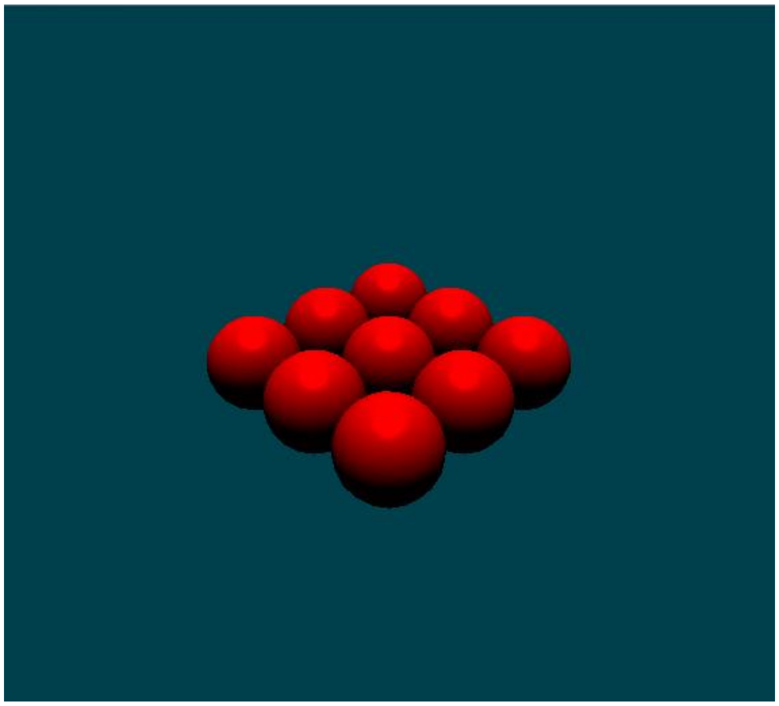
## Sample output



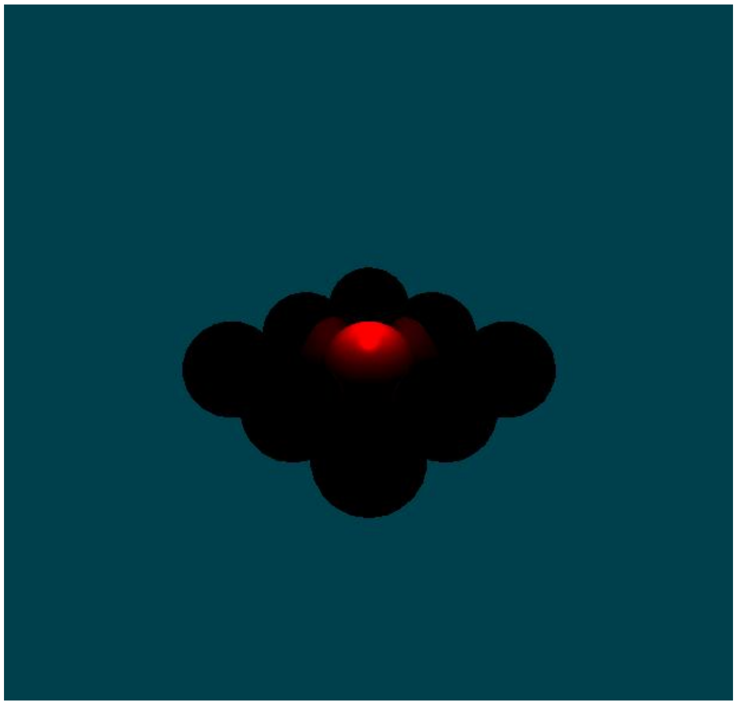
```
view
eye 3. 3. 2.
center 0. 0. 0.
viewup 0. 0. 1.

light
type directional ambient .1 .1 .1
color 1. 1. 1.  position 3. 3. 2.

light
type directional ambient .2 .2 .2
color 1. 1. 1. position 0. 0. 1.

light
type directional ambient .3 .3 .3
color 1 1 1 position 1 0 0

object OBJfiles/axis.obj
shader BLINN_PHONG
```



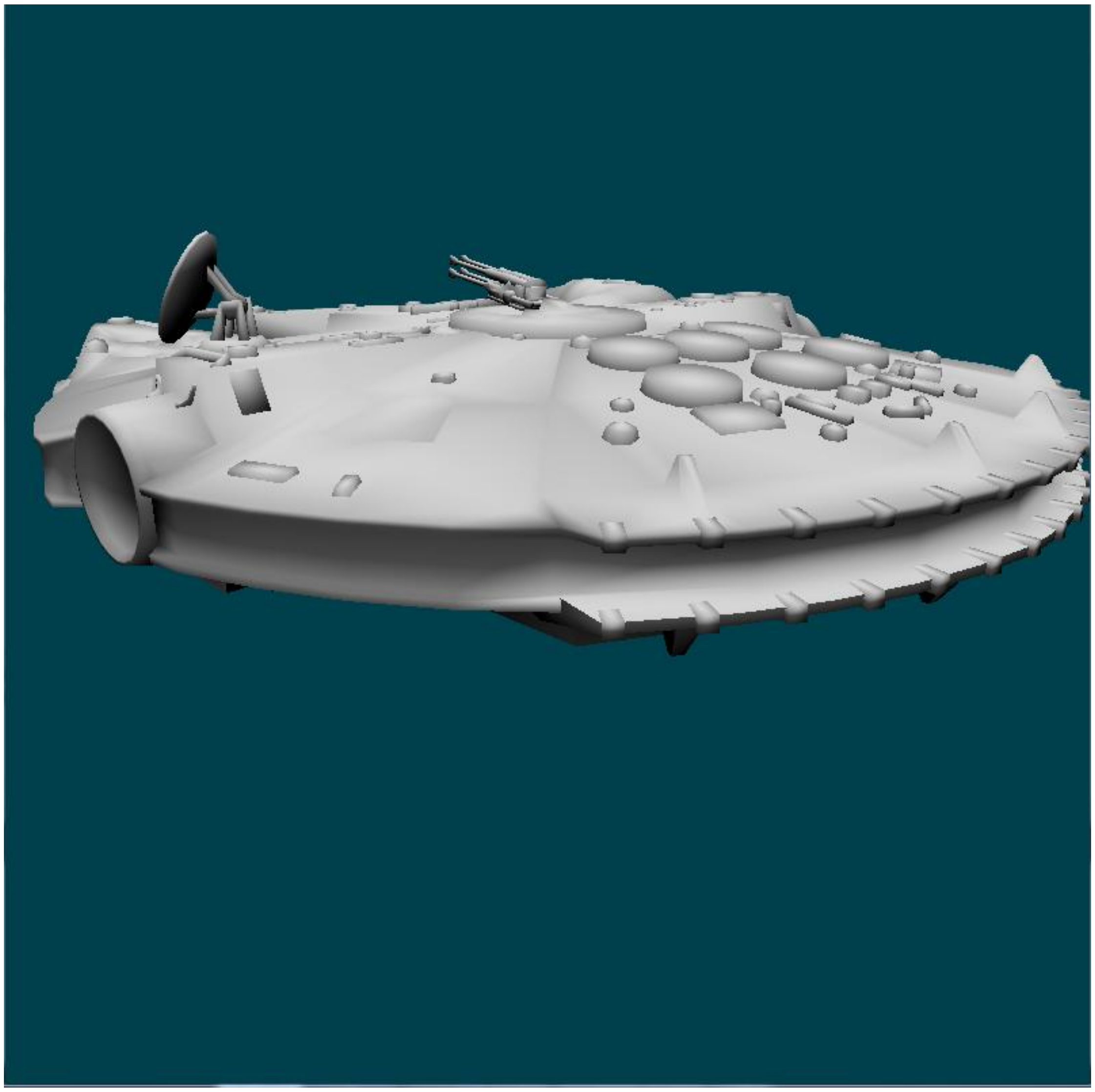
```
view
eye 150. 150. 75.
center 0. 0. 0.
viewup 0. 0. 1.

light
type directional ambient .1 .1 .1
color 1. 1. 1.  position 0. 0. 1.

light
type local
ambient .1 0 0 color 1. 0 0 position
25. 25. 40.
constAtt 0.01 linearAtt 0.01 quadAtt
.002

object OBJfiles/teapot.obj
shader BLINN_PHONG
rx 90.
```



```
view
eye 10. 10. 7. center 3. 3. 1.
viewup 0. 0. 1.

light
type directional ambient .1 .1 .1
color 1. 1. 1.  position 0. 0. 1.

object OBJfiles/sphere2.obj

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 2 0 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 4 0 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 0 2 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 2 2 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 4 2 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 0 4 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 2 4 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 4 4 0
```



```
view
eye 10. 10. 7.
center 3. 3. 1.
viewup 0. 0. 1.

light
type spot
ambient .1 .1 .1 color 1. 1. 1. position 2. 2. 4.
constAtt 0.2 linearAtt 0.2 quadAtt .002
coneDirection 0. 0. -1.
spotCosCutoff .93 spotExponent 25.

object OBJfiles/sphere2.obj

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 2 0 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 4 0 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 0 2 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 2 2 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 4 2 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 0 4 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 2 4 0

object OBJfiles/sphere2.obj
shader BLINN_PHONG
t 4 4 0
```



```
view
eye 400 400 200
center 0. 0. 0.
viewup 0. 0. 1.

light
type directional ambient .1 .1 .1
color 1. 1. 1.  position 0. 0. 1.

light
type directional ambient .1 .1 .1
color .5 .5 .5    position 4 4 3

object OBJfiles/Millennium_Falcon.obj
shader BLINN_PHONG
rx 90.
```

## Submission Instructions

Submit your files from the host **lectura.cs.arizona.edu** using the command
**turnin  cs433s17-assg2  [files]**

## Assignment Advice

1. Start early.
2. Do your own work.
3. Check piazza regularly.