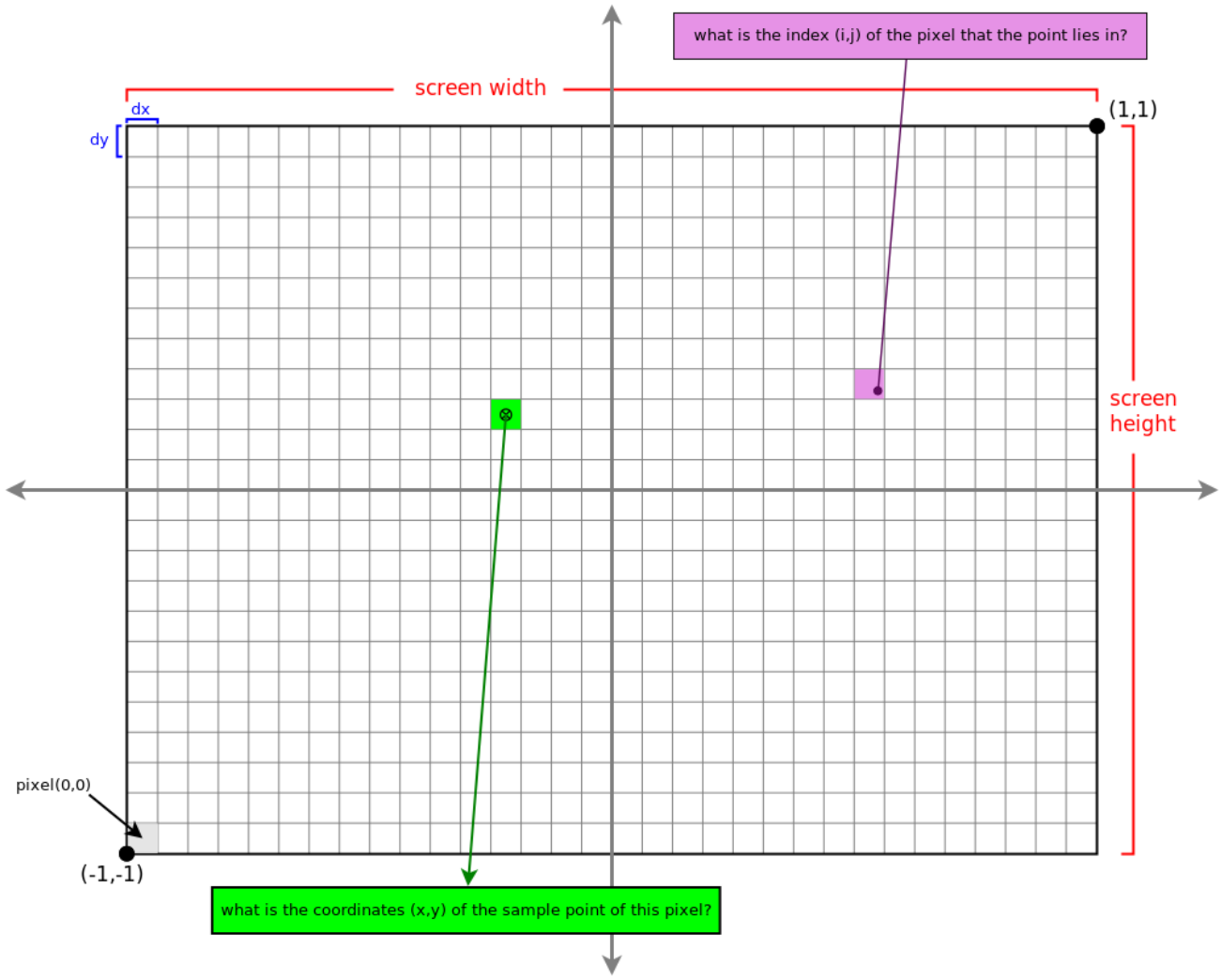
Name: **SID:** **LAB Section:**

Lab 6 - Part 1: Pixels and Triangle Rasterization

1. Coordinate space conversions. In OpenGL, the coordinates of a vertex in model-view space can be converted to NPC coordinates, after multiplying it with model-view and projection matrices and dividing it with w. The NPC coordinate frame can be visualized as a unit box covering the coordinates from -1 to 1 in all 3 dimensions. For this exercise assume a 2D space, where the NPC coordinates, x and y, are in the range of [-1,1].

In the rasterization process, the color of each pixel is decided. The pixel space can be represented by a rectangular grid with size: width x height. In this process, the vertices should be first converted to NPC space and then to pixel coordinates in order to find out which pixels lies within a given shape.

An overlapping drawing of NPC space and pixel (screen) space is given below. Answer the questions by using this figure as a reference.



Assume a 2D point,P, lies at NPC coordinates (x,y), where x,y is in range [-1,1]

a. Find the column index, i, and row index, j, of the cell (pixel) that contains the point P.

(EQ.1) i = _____(in terms of width and x)

(EQ.2) j = _____(in terms of height and y)

b. Find the coordinate, (fi,fj), of P in the pixel space, where fi is the x-coordinate and fj is the y-coordinate of P in pixel space. This is different from above as fi and fj are floating point numbers, since the pixel space is a continuous space with the same size as the pixel grid, and the origin of the coordinate frame is the center of the pixel (cell) with index (0,0).

(EQ.3) fi =_____(in terms of width and x)

(EQ.4) fj =_____(in terms of height and y)

Assume a pixel, I, with pixel index (i,j), where i is the column index and j is the row index of I.

c. Find the NPC space coordinates, (x,y), of the pixel I. Where x, and y lies within range [0,1].

(EQ.5) x = _____(in terms of width and i)

(EQ.6) y = _____(in terms of height and j)

2. Barycentric coordinates. The (signed) area of triangles can be used to calculate the barycentric coordinate of a point. Given the point P and triangle ABC fill the equations and write the barycentric coordinates using areas of triangles.

(EQ.7)

$$\alpha = \frac{area(PBC)}{area(ABC)} \qquad \beta = \_\_\_\_\_ \qquad \gamma = \_\_\_\_\_$$

There is an easy equation you can use to calculate the area of a triangle using only the coordinates of each vertex. It will be very useful since it will give you negative values if the triangle has a certain orientation. For a triable with vertices a,b,c, the equation is given by

(EQ.8) $$\text{area(abc)}=0.5\,(a_x b_y - a_y b_x + b_x c_y - b_y c_x + c_x a_y - c_y a_x).$$

Since we are interested only in the ratio between areas, you can drop 0.5 and reorganize the equation as

(EQ.9) $$\text{area(abc)}=a_x\,(b_y - c_y) + a_y\,(c_x - b_x) + (b_x c_y - b_y c_x).$$

NOTE: The order of vertices is important as it might result in change of sign.

Name: **SID:** **LAB Section:**

Lab 6 - Part 2: OpenGL Functions

1. Read the assignment page, and try to understand the general outline and the requirements

2. Briefly, in this assignment you will be implementing some of the OpenGL commands. The program in the skeleton code, reads a sequence of commands from an input file, e.g. ./tests/00.txt, and calls the functions, that you will implement in the minigl.{h,cpp} followed by a mglReadPixels call. For example, glBegin(GL_QUADS) will invoke mglBegin(MGL_QUADS).

   You are required to implement a subset of OpenGL functions, by reading the OpenGL documentation and following the provided tips.

3. Before beginning implementation, read the OpenGL documentation and describe what each function below is doing and what the inputs to these functions refer to.

The index page of the documentation can be found in the following address: (google 'opengl 2.0 reference')
https://www.khronos.org/registry/OpenGL-Refpages/es2.0/

Note: **In this assignment, your functions should match the OpenGL documentation EXACTLY!**

Important:	MathML markup language is used in the reference pages. Some browsers, such as Google Chrome, do not support this feature. Please use a browser that supports MathML (such as Firefox) especially when viewing matrices in the docs.

a. glBegin:

input:

b. glEnd:

c. glVertex2:

inputs:

d. glVertex3:

inputs:

e. glColor:

inputs:

f. glReadPixels:

inputs:

Important:	In the rest of the assignment, please do not forget to read the documentation of a function before implementing it.

Here are 2 example OpenGL command sequences that draw a cube and a pyramid. Go through the code and try to get an understanding of the commands.

Render a color-cube consisting of 6 quads with different colors

```
// Begin drawing the color cube with 6 quads
glBegin(GL_QUADS);
// Top face (y = 1.0f)
   glColor3f(0.0f, 1.0f, 0.0f);  // Set color to Green
   glVertex3f( 1.0f, 1.0f, -1.0f); // Add vertex
   glVertex3f(-1.0f, 1.0f, -1.0f); // Add vertex
   glVertex3f(-1.0f, 1.0f,  1.0f); // Add vertex
   glVertex3f( 1.0f, 1.0f,  1.0f); // Add vertex
   // Bottom face (y = -1.0f)
   glColor3f(1.0f, 0.5f, 0.0f);  // Set color to Orange
   glVertex3f( 1.0f, -1.0f,  1.0f); // Add vertex
   glVertex3f(-1.0f, -1.0f,  1.0f); // Add vertex
   glVertex3f(-1.0f, -1.0f, -1.0f); // Add vertex
   glVertex3f( 1.0f, -1.0f, -1.0f); // Add vertex
   // Front face  (z = 1.0f)
   glColor3f(1.0f, 0.0f, 0.0f);  // Set color to Red
   glVertex3f( 1.0f,  1.0f, 1.0f); // Add vertex
   glVertex3f(-1.0f,  1.0f, 1.0f); // Add vertex
   glVertex3f(-1.0f, -1.0f, 1.0f); // Add vertex
   glVertex3f( 1.0f, -1.0f, 1.0f); // Add vertex
   // Back face (z = -1.0f)
   glColor3f(1.0f, 1.0f, 0.0f);  // Set color to Yellow
   glVertex3f( 1.0f, -1.0f, -1.0f); // Add vertex
   glVertex3f(-1.0f, -1.0f, -1.0f); // Add vertex
   glVertex3f(-1.0f,  1.0f, -1.0f); // Add vertex
   glVertex3f( 1.0f,  1.0f, -1.0f); // Add vertex
   // Left face (x = -1.0f)
   glColor3f(0.0f, 0.0f, 1.0f);  // Set color to Blue
   glVertex3f(-1.0f,  1.0f,  1.0f); // Add vertex
   glVertex3f(-1.0f,  1.0f, -1.0f); // Add vertex
   glVertex3f(-1.0f, -1.0f, -1.0f); // Add vertex
   glVertex3f(-1.0f, -1.0f,  1.0f); // Add vertex
   // Right face (x = 1.0f)
   glColor3f(1.0f, 0.0f, 1.0f);  // Magenta
   glVertex3f(1.0f,  1.0f, -1.0f); // Add vertex
   glVertex3f(1.0f,  1.0f,  1.0f); // Add vertex
   glVertex3f(1.0f, -1.0f,  1.0f); // Add vertex
   glVertex3f(1.0f, -1.0f, -1.0f); // Add vertex
glEnd();  // End of drawing color-cube
```

Render a pyramid consists of 4 triangles

```
// Begin drawing the pyramid with 4 triangles
glBegin(GL_TRIANGLES);
   // Front
   glColor3f(1.0f, 0.0f, 0.0f);  // Red
   glVertex3f( 0.0f, 1.0f, 0.0f); // Add vertex
   glColor3f(0.0f, 1.0f, 0.0f);  // Green
   glVertex3f(-1.0f, -1.0f, 1.0f); // Add vertex
   glColor3f(0.0f, 0.0f, 1.0f);  // Blue
   glVertex3f(1.0f, -1.0f, 1.0f); // Add vertex
   // Right
   glColor3f(1.0f, 0.0f, 0.0f);  // Red
   glVertex3f(0.0f, 1.0f, 0.0f); // Add vertex
   glColor3f(0.0f, 0.0f, 1.0f);  // Blue
   glVertex3f(1.0f, -1.0f, 1.0f); // Add vertex
   glColor3f(0.0f, 1.0f, 0.0f);  // Green
   glVertex3f(1.0f, -1.0f, -1.0f); // Add vertex
    // Back
   glColor3f(1.0f, 0.0f, 0.0f);  // Red
   glVertex3f(0.0f, 1.0f, 0.0f); // Add vertex
   glColor3f(0.0f, 1.0f, 0.0f);  // Green
   glVertex3f(1.0f, -1.0f, -1.0f); // Add vertex
   glColor3f(0.0f, 0.0f, 1.0f);  // Blue
   glVertex3f(-1.0f, -1.0f, -1.0f); // Add vertex
   // Left
   glColor3f(1.0f,0.0f,0.0f);    // Red
   glVertex3f( 0.0f, 1.0f, 0.0f); // Add vertex
   glColor3f(0.0f,0.0f,1.0f);    // Blue
   glVertex3f(-1.0f,-1.0f,-1.0f); // Add vertex
   glColor3f(0.0f,1.0f,0.0f);    // Green
   glVertex3f(-1.0f,-1.0f, 1.0f); // Add vertex
glEnd();   // Done drawing the pyramid
```

NOTE: Feel free to copy these codes to a text file and use it as an input file for the program

Name: **SID:** **LAB Section:**

Lab 6 - Part 3: Starting Assignment 2

Download the skeleton code from the assignment webpage.
Compile and run with a few test files (with -g option) to get a feel to the program.

Extract Code: tar -xzf proj-gl-files.tar.gz
Compile: scons
Run: ./minigl -g -i ./tests/*N*.txt where *N=00,01,...28*
Grade Preview Tests: ./grading-script.sh ./tests

Follow the guidelines below and start implementing the functions.

Note: All the implementation should be done in minigl.cpp; you do not need to change any file other than minigl.cpp in this assignment.

Don't forget to compile after every step!
You won't see any output till you finish everything in this lab.

You may not be allowed to leave the lab until you pass the first 6 tests, for full credit.

1. mglColor sets the current/active color.

a. Create a variable that stores a color as a 3d vector

_____

b. Copy the code above to minigl.cpp (as a global variable). Find and implement the mglColor function in minigl.cpp

2. **glVertex2** and **glVertex3** adds vertices to a vertex list. Therefore, we would need a Vertex structure and a variable that stores a list of vertices.

a. Create a structure that stores a 4d position and a color.

```
struct Vertex
{
      _____

      _____

};
```

b. Create a variable that stores a list of vertices.

_____

c. Copy the code above to minigl.cpp. You may add a constructor for vertex class or use initializer lists to construct it when needed.

d. Locate mglVertex3 function in minigl.cpp. In this function, you will need to create a vertex with a 4d position (x,y,z,w) and a color. Use the current color variable (1.a) for the color and the input parameters to create the 4d vector setting its w to 1, e.g. vec4(x,y,z,1). Add the created vertex to the vertex list (2.b).
Note: Later on in the assignment, you will have to use modelview and projection matrices. Make sure to multiply the vertex with the (current) modelview and projection matrices in the correct order before adding it to the list.

e. A 2D vector can be implemented as a 3d vector with z=0. Locate and implement the mglVertex2 function. Hint: you may utilize the mglVertex3 function.

3. **glBegin** starts a list of vertices for primitives. The type of the primitives is given as an input to this function. In this assignment, you will need to implement only triangles and quads indicated by MGL_TRIANGLES and MGL_QUADS respectively.

a. Create a variable that keeps the current type of primitive (see typedefs in minigl.h and the glBegin parameter to decide the 'type' of the variable).

_____

b. Copy the code above to minigl.cpp (as a global variable).

c. Locate the mglBegin function in minigl.cpp, and set the current primitive type according to the input.

4. **glEnd** marks the end of acceptance of vertices for the current primitive(s). The collected vertices in-between the glBegin and glEnd are converted into a list of primitives (triangles). Given N vertices, if the current primitive mode is MGL_TRIANGLES, a triangle should be created for each triplet of vertices and added to a list of triangles.

a. Create a structure that stores the 3 vertices of a triangle, e.g. A,B and C.

```
struct Triangle
{
      _____

};
```

b. Create a variable that stores a list of triangles.

_____

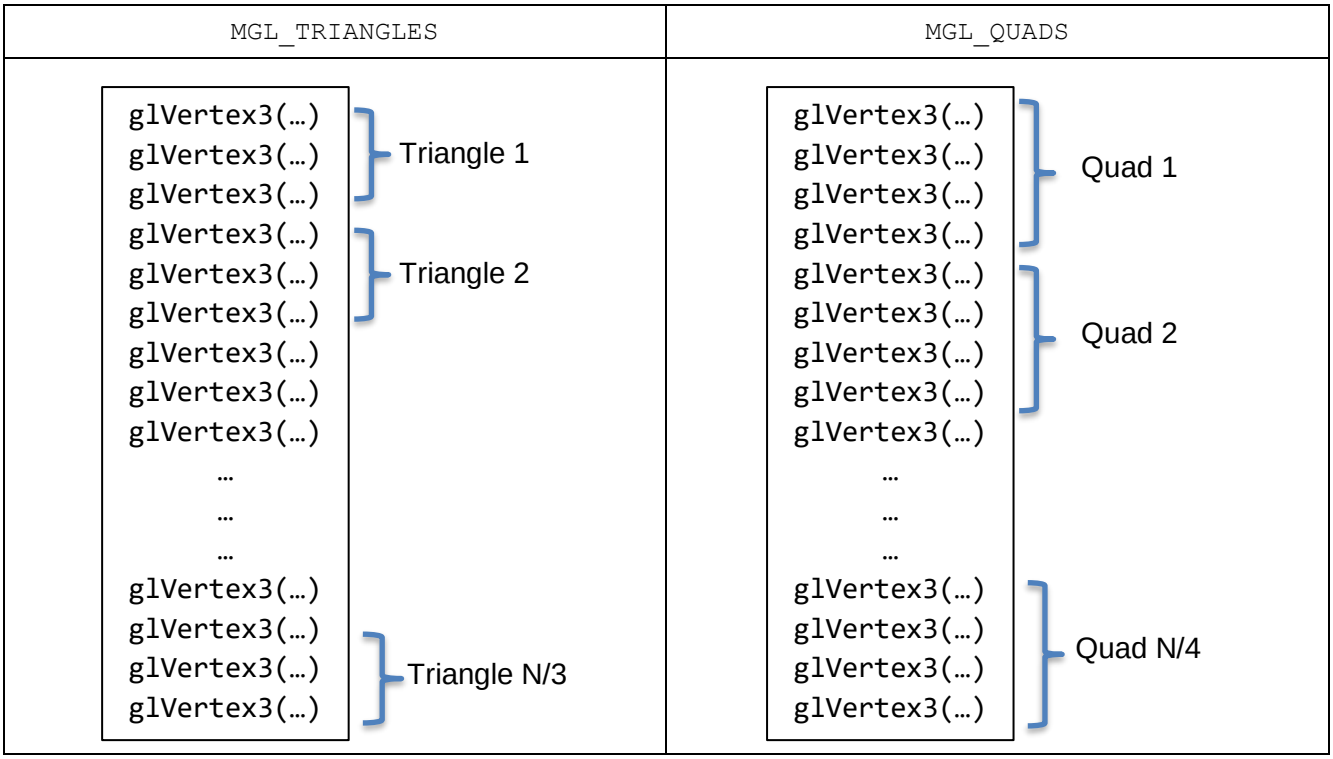


Figure: Vertex decomposition for different primitives.

c. Copy the code above to minigl.cpp (globally). Locate mglEnd and begin implementing the function: if the current primitive type (3.a) is MGL_TRIANGLES, then iterate through the vertex list (3.b) and create a triangle from each triplet of vertices, and add to the triangle list (4.b).

d. A quad can be represented with 2 triangles. See the figure below for an example. In mglEnd function, if the current primitive type is MGL_QUADS, iterate through the vertex list (3.b) and create two triangles for each quad, from each quadruplet of vertices, and add to the triangle list.

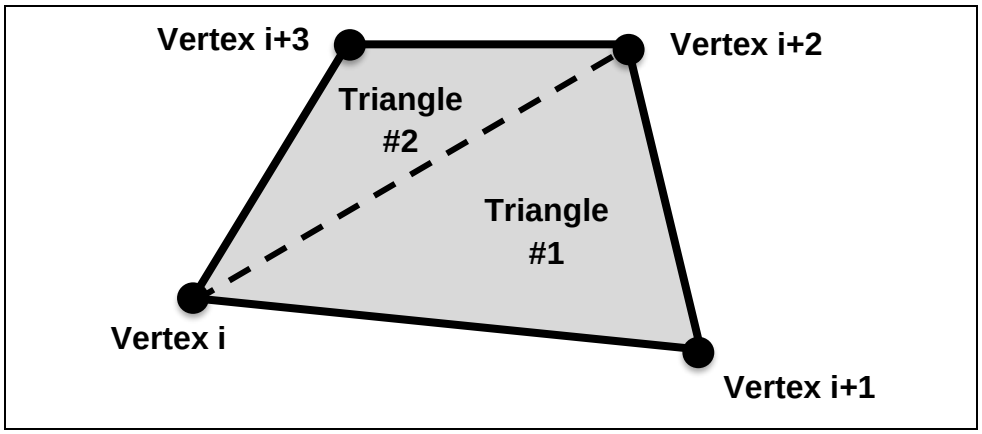


Figure: A quad can be represented with 2 triangles.

e. Make sure to clean the vertices list at the end of mglEnd function.

5. glReadPixel function is used to set the color of each pixel in the OpenGL window. The inputs width and height specify the size of the window, and data is the pointer to the readily allocated output 1D output array, *data*. In mglReadPixel function, you will need to rasterize the triangles onto the pixels by setting each pixel's color.

The *data* array is a row-major ordered 1D storage for the color of the pixels. In order to set the color of a pixel. you need to convert the color from (float) [0,1] to (int) [0,255] range and use Make_Pixel function defined in minigl.h, that will convert it to 32-bit color representation.
e.g. The following line of code set the color pixel (i,j) to red (1,0,0).

```
data[i+j*width] = Make_Pixel(255,0,0) //(0.4,0.4,0.4) x 255 = (102,102,102)
```

To set the color to gray (0.4,0.4,0.4) =>

```
data[i+j*width] = Make_Pixel(102,102,102) //(0.4,0.4,0.4) x 255 = (102,102,102)
```

a. In minigl.cpp, create a helper function, Rasterize_Triangle, that gets a Triangle, width, height, and data as its parameters and rasterizes the triangle on the screen by setting the colors in data:

```
void Rasterize_Triangle(const Triangle& tri, int width, int height, MGLpixel* data)
```

In this function:

- Calculate the pixel coordinates of the vertices of the triangle (EQ. 3,4)
- For each pixel I=(i,j) in the screen, with size: width x height
    - Calculate the barycentric coordinate of I, by using it with the pixel coordinates of the triangle vertices (EQ. 7,9).
      Suggestion: Add another helper function that gets 3 points and returns the area, for a cleaner code.
    - Using barycentric coordinates, decide if the pixel is inside the triangle. If so, color the pixel
      (You may use the current color, color of the first pixel or white (255,255,255) for all the interior pixels for now.)
      Do not color with black if the pixel is outside as it might overwrite prior triangles.

You can visit the class notes on triangle rasterization for further explanation and additional optimizations (such as iterating only on bounding boxes – use EQ.1,2 to do so).

b. Locate mglReadPixels function and start implementing it.

- Fill the whole pixel data with black using Make_Pixel(0,0,0)

- For each triangle in the triangle list call the Rasterize_Triangle function to rasterize it on the 'data'.
- Clear the triangle list.

6. glOrtho multiplies the current matrix with the orthographic matrix given in its documentation page:
https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glOrtho.xml
For the first checkpoint, you do not need to multiply it but rather set it.

In addition to the vec class the skeleton code also includes a mat class for matrix storage and operations.

Matrix Class and Ordering

Basics:
The matrix class (math) stores an array of values (16 for mat4) in column major order (as in opengl matrices).

- The matrix and vector classes provide the basic arithmetic operations such as multiplication
- You can access the elements of vector by using [] operator. e.g. v[0] is the x-coordinate, v[1] is the y-coordinate, etc.
- You can access the elements of matrix by [] operator. e.g. m[i,j] would access the element at ith row and jth column. (matrix is zero indexed).

Ordering and Initialization:

The 4x4 matrix class (mat4) stores the values in column-major order, just like OpenGL matrices. When the matrix is created, the values are allocated but not initialized. use make_zero() function to initialize it to 0.
When initializing the matrix with brace-enclosed initializer lists, make sure you use the correct order.

Tip: enter it like its transpose

The code :

```
mat4 A={{a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15}};
```

would initialize matrix A as:

$$A = \begin{bmatrix} a0 & a4 & a8 & a12 \\ a1 & a5 & a9 & a13 \\ a2 & a6 & a10 & a14 \\ a3 & a7 & a11 & a15 \end{bmatrix}$$

Warning: When using brace-enclosed initializer lists, make sure all 'a' values are MGLfloat, e.g., entering 1.0 as one of the values, would not work since it is a double, you can use 1.0f instead.

a. Create a 4x4 matrix (mat4) variable in minigl.cpp

```
mat4 projection;
```

b. In the mglOrtho function, set the projection matrix with the matrix provided in the documentation.

c. In the mglVertex3/2 functions left-multiply the Vertex with the projection matrix before adding it to the vertex list. Note: This should be changed to projection x modelview x vertex in the upcoming checkpoints

Notes for checkpoints 2 and onwards:

For all of the matrix operations, please refer to the opengl documentation
**In order to view matrices and equations properly, you should use firefox or a browser that supports MathML.*

> **Note:** **The matrices generated by your functions should match the opengl documentation EXACTLY!**

Here are brief descriptions and suggestions for some functions:

mglMatrixMode: Sets the current matrix mode. Create a global variable that stores the current mode then set it in this function.

mglFrustum: *very similar to glOrtho, create the perspective projection matrix as in docs and multiply (or set for now) with the active matrix. Additionally, at the beginning of* ***Rasterize_Triangle*** *function, you will need to divide the x,y and z of each vertex with its w to find the NPC coordinates.*

mglColor:

- Set the current color with the parameters of this function.
- Modify **mglReadPixel** so that you use the color of the first vertex of a triangle in Make_Pixel rather than (255,255,255). *Tip: Make sure that you convert each vertex color,(r,g,b), from [0,1] domain to [0,255] before using them in Make_Pixel.*

Matrix Stacks:

First of all, if you do not feel familiar about matrix stacks in opengl, please read " Manipulating the Matrix Stacks" section of **this guide**:
http://www.glprogramming.com/red/chapter03.html#name6

- Before implementing any new function, you need to change the projection and modelview matrices to projection and modelview stacks. *I suggest using std::vector for stacks, where back() function gives the top of the stack.*
- When declaring/initializing these stacks make sure they have 1 element/mat4 in them. *i.e. use an appropriate constructor. The values inside the initial matrices does not matter.*
- I also suggest adding a helper function that returns a reference to the top of the active stack (based on the current matrix mode); e.g. *top_of_active_matrix_stack()*. So that, we can avoid having if/else structures in matrix modifying functions.
- Modify all the already-implemented functions that changes matrices such as **mglFrustum**, **mglOrtho**, **mglLoadIdentity**, etc., so that they work with the top of the active matrix stack.

- In **mglVertex** functions, we are still multiplying each vertex position with both projection and modelview matrices, but this time just using the top of the corresponding stack. *Tip: the mat and vec classes provide multiplication operators.*
- Run tests and make sure you still pass tests 00-05.
- **mglPushMatrix**, **mglPopMatrix**: do the push or pop operations on the active stack. Implementing these would not pass another test for now. *Note: These are the only functions that should do the push and pop operations.*

mglScale, **mglTranslate**, **mglRotate**:

- Follow the description in opengl documentation for each function.
- This is up to you, but you can implement and use the **mglMultMatrix** and use the pointer to the head of your multiplier matrix by accessing it with `&multiplier.values[0]`

Tips on mglRotate:

1. you should normalize the (x,y,z) vector.

2. The function uses degrees [0,360] as input, however cos and sin functions use radians.

Further notes may be provided on Piazza...