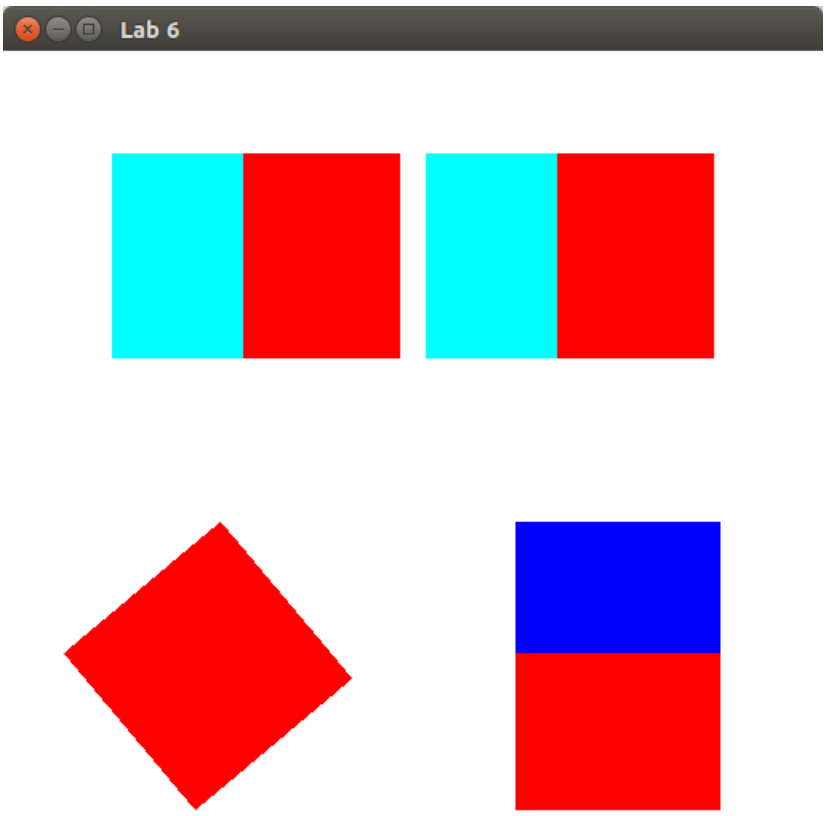
Lab 6: Multiple Objects Independent Transformations

The purpose of this lab is for you to learn how to draw multiple objects where each object has its own transformation. What are you going to do is to four cubes where the size each cube is $0.5 \times 0.5 \times 0.5$. For each cube, it should have 6 different colors (one per face). The specification of each cube are as follow:

- **Cube 1**: The center of mass of the cube located at $(-0.5, 0.5, 0.0)$
- **Cube 2**: The center of mass of the cube located at $(0.5, 0.5, 0.0)$
- **Cube 3**: The center of mass of the cube located at $(-0.5, -0.5, 0.0)$ and it spins continuously about z-axis and its fix point of rotation is at its center of mass $(-0.5, -0.5, 0.0)$.
- **Cube 4**: The center of mass of the cube located at $(0.5, -0.5, 0.0)$ and it spins continuously about x-axis and its fix point of rotation is at its center of mass $(0.5, -0.5, 0.0)$.

For Cube 1 and Cube 2, consider them as one single object. Let's call the object **twin-cube**. For this twin-cube, it should spins continuously about y-axis and its fix point of rotation is the midpoint between the center of masses of those two cubes. In other words, the fix point of rotation is $(0.0, 0.5, 0.0)$. Note that you are going to use the `idle()` function to make all your cubes spin continuously.



Shader Program

For this project, you are going to use the same shader programs as in your lab 4 (CTM):

vshader.glsl

```
#version 120

attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 color;

uniform mat4 ctm;

int main()
{
    gl_Position = ctm * vPosition;
    color = vColor;
}
```

fshader.glsl

```
#version 120

varying vec4 color;

int main()
{
    gl_FragColor = color;
}
```

Building the World

For simplicity, you should have a cube in your model frame of size $1 \times 1 \times 1$

```
vec4 cube_vertices[36] = {...};
```

This can be manually defined vertices instead of computer generated. However, if you want to write a function to generate these 36 vertices, go ahead.

Now, we are going to practice building a 3D world in two different ways:

1. Creating an object by transforming before sending its vertices into the graphic pipeline
2. Sending a model as is into the graphics pipeline and transform it later

So, for this lab, create an array of vertices of size 108. Let's call this array of vertices `vertices`. The first 72-vertices will be for the twin-cube and the last 36-vertices is for the simple cube.

For the twin-cube, scale and transform each cube to its final position by copying transformed vertices to `vertices`. Your code should look similar to the following:

```
vec4 cube_vertices[36] = {...};
vec4 cube_colors[36] = {...};
int i, v_index = 0;

for(i = 0; i < 36; i++)
{
        vertices[v_index] = m4v4_multiplication(temp_tr1, cube_vertices[i]);
        colors[v_index] = cube_colors[i];
        v_index++;
}

for(i = 0; i < 36; i++)
{
        vertices[v_index] = m4v4_multiplication(temp_tr2, cube_vertices[i]);
        colors[v_index] = cube_colors[i];
        v_index++;
}
```

where `temp_tr1` is the transformation matrix that scales the $1 \times 1 \times 1$ cube to $0.5 \times 0.5 \times 0.5$ and translate it so that its center of mass is at $(-0.5, 0.5, 0.0)$. Similarly, `temp_tr2` is the transformation matrix that scales the $1 \times 1 \times 1$ cube to $0.5 \times 0.5 \times 0.5$ and translate it so that its center of mass is at $(0.5, 0.5, 0.0)$.

For the two cubes, we are going to send the `cube_vertices` as is into the graphics pipeline. Thus, we simply perform:

```
for(i = 0; i < 36; i++)
{
        vertices[v_index] = cube_vertices[i];
        colors[v_index] = cube_colors[i];
        v_index++;
}
```

Note that your code may be different than mine depending on your implementation of vectors, matrices, and their operations. Next, transfer all 108 vertices and their colors into the graphic pipeline as usual.

Display

TO display these objects, it will be a little bit different than what we have done before. Since we have three objects, twin-cube and two cubes. Each has its own attribute (the location of its center of mass, current degree of rotation, current transformation matrix), we should maintain them separately as global variables:

```
vec4 twin-cube_location = {0.0, 0.5, 0.0, 1.0};
vec4 left-cube_location = {-0.5, -0.5, 0.0, 1.0};
vec4 right-cube_location = {0.5, -0.5, 0.0, 1.0};
```

```
GLfloat twin-cube_degree = 0.0, left-cube_degree = 0.0, right-cube_degree = 0.0;
mat4 twin-cube_ctm = m4_identity();
mat4 left-cube_ctm = m4_identity();
mat4 right-cube_ctm = m4_identity();
```

where `m4_identity()` returns a 4×4 identity matrix. When we draw these objects, we are going to draw them one-by-one. For each object, we are going to draw it based on its attribute as well as how its data (vertices and colors) are stored in the graphics pipeline. Since we are going to transform each object differently, in the display function, we have to tell the OpenGL to draw three times as shown below:

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glUniformMatrix4fv(ctm_location, 1, GL_FALSE, (GLfloat *) &twin-cube_ctm);
    glDrawArrays(GL_TRIANGLES, 0, 72);

    glUniformMatrix4fv(ctm_location, 1, GL_FALSE, (GLfloat *) &left-cube_ctm);
    glDrawArrays(GL_TRIANGLES, 72, 36);

    glUniformMatrix4fv(ctm_location, 1, GL_FALSE, (GLfloat *) &right-cube_ctm);
    glDrawArrays(GL_TRIANGLES, 72, 36);
}
```

Note that we draw the left-cube and the right-cube from the same set of vertices in the graphics pipeline but two different CTMs are used, `left_cube_ctm` and `right_cube_ctm`.

HINT

- Obviously the `idle()` function will have to change the degree of rotation of each object every time it is called. So, increase it by a very small degree (e.g., 0.05) and then calculate the new CTM for each object. Do not forget to call the `glutPostRedisplay()` function at the end.
- Since the fixed point of rotation of the twin-cube is not at the origin, you must translate its fixed point of rotation to the origin first, rotate it, and then translate it back.
- Since vertices of two cubes that we are going to use are the $1 \times 1 \times 1$ cube where the center of mass is at the origin. To draw each of these cube, first, we need to scale it to the right size ($0.5 \times 0.5 \times 0.5$), rotate it, and then translate it to is final position.

Due Date and Submission

For the due date, please check the lab in the CourseWeb. Submit your program (zip all files) to the CourseWeb under this lab by the due date (demo date). You also need to demo this lab in front of the TA to verify that it works correctly. **No late submission will be accepted**.