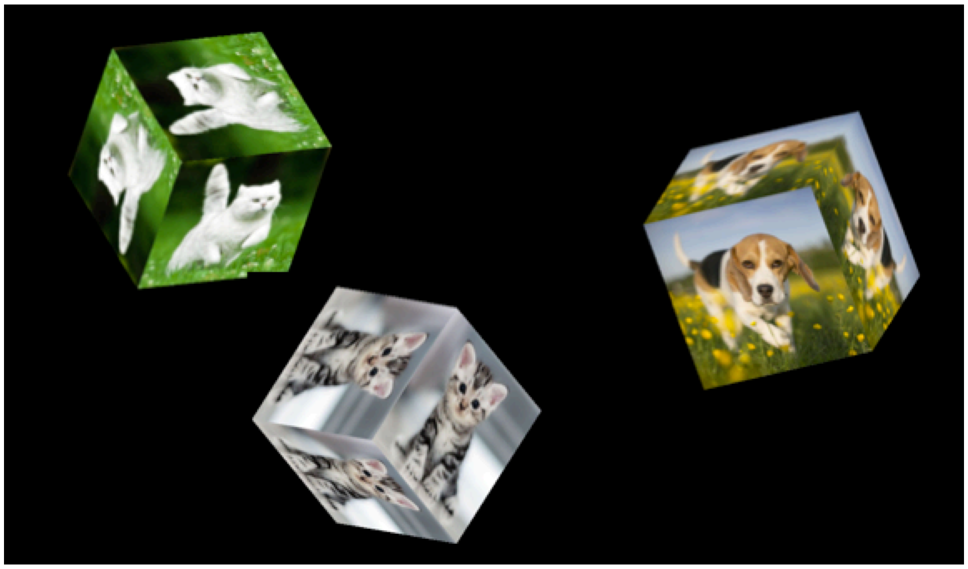
CSCE 4813 – Computer Graphics
Programming Project 5
Due Friday 04/17/2020

1. Problem Statement:

The goal of this project is to create an OpenGL program that uses texture mapping to display images on geometric objects. You will be given a collection of cat and dog images, and your task will be to map a selection of these images onto the faces of cubes as they "fall" from the top of the screen to the bottom of the screen. This way, your program will look like it is "raining cats and dogs". Your project has the following tasks:



Task 1: Reading Texture Images

Your first task is to read and store color information from cat and dog images. You can use the im_color class from the previous assignment to open and read a JPEG image into an im_color object. As you know, the im_color object contains three separate im_short objects that contain the R, G, and B images.

Next, you have to copy this RGB information into an OpenGL texture array, where pixels are stored in "row major order" and the color of each pixel is stored in an "interleaved RGB format" where bytes 0,1,2 store the R,G,B values for the first pixel, bytes 3,4,5 store the R,G,B values for the second pixel, and so on.

The sample images you have been provided have all been cropped and interpolated to be 512x512 pixels. Hence, your OpenGL texture arrays should be declared as unsigned byte arrays that are 512x512x3 long. If you look at the sample program "texture4.cpp" you will see code for reading a JPEG image and copying pixels from 2D im_color arrays into the 1D texture array.

Task 2: Model Creation

Your next task is to create a data structure that represents the falling cubes. To do this, you need to keep track of the following information for each of the N cubes in your "raining cats and dogs" simulation:

- The cube position (Px, Py, Pz)
- The cube velocity (Vx, Vy, Vz)
- The cube rotation angles (Ax, Ay, Az)
- The cube radius R
- The OpenGL texture array for this cube

Your "init" function should initialize a global data structure containing information for the N cubes in your program using random values for the cube position, velocity, rotation angles, and radius. You should take care that the cube position coordinates are within the viewing region specified by glOrtho() and that the Py position is near the top of the screen, and the Vy is negative so the cube moves downward. To initialize the texture array, you can randomly choose one of the cat or dog images.

Task 3: Model Display

Your next task is to write an OpenGL program that loops over the N cubes in your scene and displays the falling cubes at the correct location and orientation. The easiest way to do this is to call OpenGL transformation functions to define the MODELVIEW matrix that rotates, translates, and scales each cube prior to display.

To start, you may want to display the six faces of the cube using GL_LINE_LOOP to verify they are drawn were expected. Once you have cube display working, you can switch to GL_POLYGON, and display each of the six faces using the texture map for that cube. To do this, you will need to "turn on" texture mapping in the "init" function. This is done with the following commands:

```
glEnable(GL_TEXTURE_2D);
glTexParameterf(GL_TEXTURE_2D, GL_GENERATE_MIPMAP, GL_TRUE);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_NEAREST);
```

The "display" callback should also be modified to specify which texture map to use to display this cube. This is done with the following:

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, xdim, ydim,
      0, GL_RGB, GL_UNSIGNED_BYTE, texture);
```

You also need to specify the texture coordinates (tx,ty) at each polygon vertex. The texture coordinates should be between [0..1] where (0,0) is the top left corner of the image, and (1,1) is the bottom right. This is done using the following function calls:

```
glTexCoord2f(tx1, ty1);
glVertex3f(x1, y1, z1);
glTexCoord2f(tx2, ty2);
glVertex3f(x2, y2, z2);
glTexCoord2f(tx3, ty3);
glVertex3f(x3, y3, z3);
glTexCoord2f(tx4, ty4);
glVertex3f(x4, y4, z4);
```

By playing around with (tx,ty) coordinates, you can display all or part of your cat image on each face of the cube. If you use texture coordinates that are outside the [0..1] range, OpenGL will "wrap around" the coordinates into the [0..1] range. This may result in multiple copies of the image being texture mapped onto the polygon.

Task 4: Animating the Model

Finally, to make your program simulate "raining cats and dogs" you will need to update the location and orientation of your cubes to make it look like they are falling downward. To do this, you will have to define an "idle" callback or a "timer" callback that loops over the N cubes in your model, and updates the (Px,Py,Pz) position using (Vx,Vy,Vz) velocity information. You can also increment the (Ax,Ay,Az) values by a small amount so the cubes will spin slowly. You do not need to change the radius R or the texture map.

2. Design:

There are several design tasks you must complete for this project. The most important decision is to select data structures to store the information about your N cubes described above. Remember, this will need to be global so you can access it everywhere in the program.

Your next big task is to decide how to initialize the location, orientation, and size of the N cubes in your simulation. To start with, you may want just one cube near the top of the screen. The hard part here will be to choose (Px,Py,Pz) values that "look nice" on the screen and do not overlap each other.

Finally, you need to display your texture mapped model in "display" and animate your model in the "idle" or "timer" callback. You have to select the (tx,ty) coordinates of each vertex carefully to get the cat/dog image to display properly. You also have to be careful you do not make the cubes move too quickly, or they will be gone before we can look at them.

3. Implementation:

This semester we will be using C++ and OpenGL to implement all of our programming projects. If you are using a Mac with Xcode installed, then you can download the src.tar file and compile the sample graphics code using the enclosed Makefile. If you are using a PC, then your best option would be to download and install a Linux VM from the department's website. The instructions for doing this are posted in README file the "Source Code" page of the class website. Once you have Linux and OpenGL installed, you can compile your graphics program using

```
Linux:
g++ -Wall project5.cpp -o project5 -lGL -lGLU –lglut
libim/libim.a jpeg/libjpeg.a

Mac:
g++ -Wall project5.cpp -o project5 -DMAC -framework OPENGL
-framework GLUT libim/libim.a jpeg/libjpeg.a
```

Remember to use incremental development and good programming style when creating your program. Choose good names for variables and constants, use proper indenting for loops and conditionals, and include clear comments in your code. Also, be sure to save backup copies of your program somewhere safe. Otherwise, you may end up retyping your whole program if something goes wrong.

4. Testing:

Test your program to check that it operates correctly for all of the requirements listed above. Also check for the error handling capabilities of the code. Try your program with several input values, and save screen shots of your output in jpeg images for inclusion in your project report.

5. Documentation:

When you have completed your C++ program, write a short report using the project report template describing what the objectives were, what you did, and the status of the program. Does it work properly for all test cases? Are there any known problems? Save this report to be submitted electronically.

6. Project Submission:

In this class, we will be using electronic project submission to make sure that all students hand their programming projects and labs on time, and to perform automatic plagiarism analysis of all programs that are submitted.

When you have completed the tasks above go to Blackboard to upload your documentation (a single docx or pdf file), and all of your C++ program files. Do NOT upload an executable version of your program.

The dates on your electronic submission will be used to verify that you met the due date above. All late projects will receive reduced credit:

- 10% off if less than 1 day late,
- 20% off if less than 2 days late,
- 30% off if less than 3 days late,
- no credit if more than 3 days late.

You will receive partial credit for all programs that compile even if they do not meet all program requirements, so handing projects in on time is highly recommended.

7. Academic Honesty Statement:

Students are expected to submit their own work on all programming projects, unless group projects have been explicitly assigned. Students are NOT allowed to distribute code to each other, or copy code from another individual or website. Students ARE allowed to use any materials on the class website, or in the textbook, or ask the instructor and/or GTAs for assistance.

This course will be using highly effective program comparison software to calculate the similarity of all programs to each other, and to homework assignments from previous semesters. Please do not be tempted to plagiarize from another student.

Violations of the policies above will be reported to the Provost's office and may result in a ZERO on the programming project, an F in the class, or suspension from the university, depending on the severity of the violation and any history of prior violations.