**Homework 2 Report**

D11315807

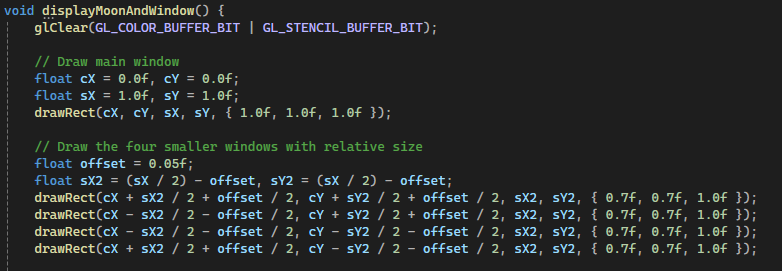
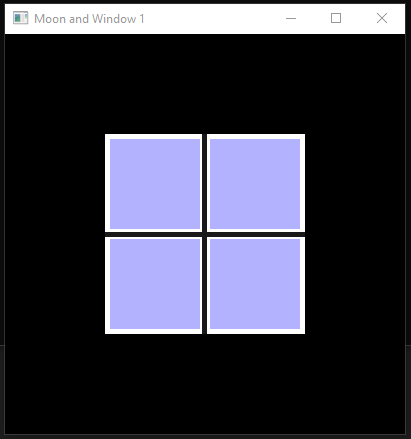
Ardiawan Bagus Harisa

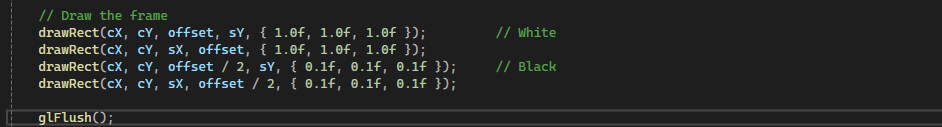
Department of CSIE

In this homework project, I apologize that my implementation may be varied from the sample instruction. I just try to be more creative.

1. **Create a window**

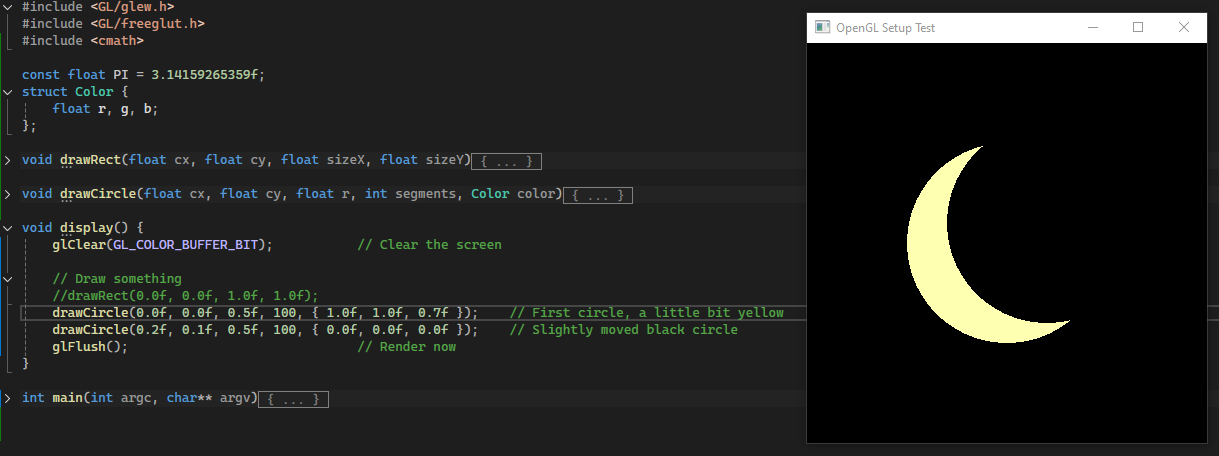
The first thing I do is create window. First, I draw the white rectangle as the main background of the window using previous drawRect() function with the size of half of the application window (200x200 px). Second, I draw four parts of glasses and use light blue color. Instead of hard-coded size, I write relative size so that it can be more dynamic. We can also change the offset (the size of the crossing frame on the window). So, it becomes like this:

** **

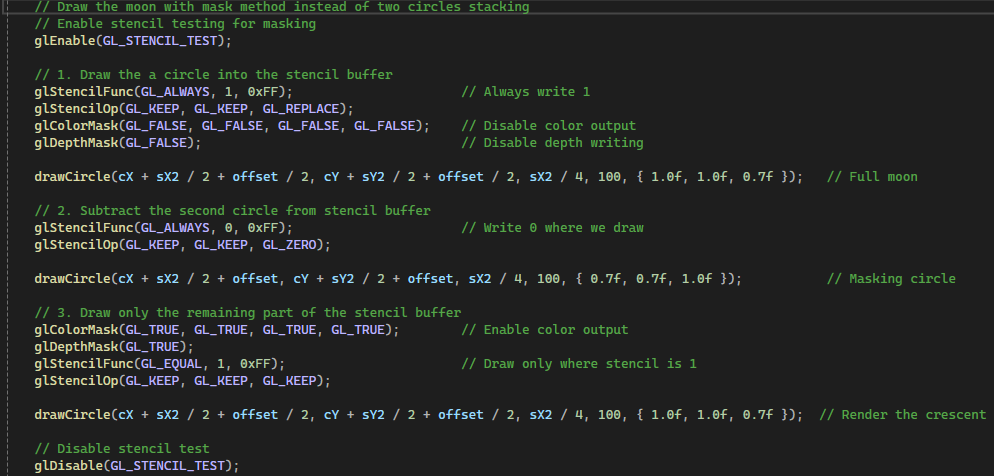
****

1. **Draw a Crescent Moon**

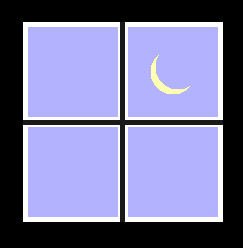
Instead of using the same technique, stacking, I use masking. The previous method is shown below, where I literally simply just draw 2 circles. However, because there are two different background colors, I must use masking. This is the previous function:



Therefore, I use the stencil buffer from OpenGL to handle the masking operation. First, I need to enable the glstenscil. Then setup some parameters regarding the color mode and the depth. This will define the output. Next, I draw the full circle and feed it to the buffer. Second, I do subtraction operation to the circle with the new slightly-moved one. The third step is to draw the remaining circle that have been subtracted.



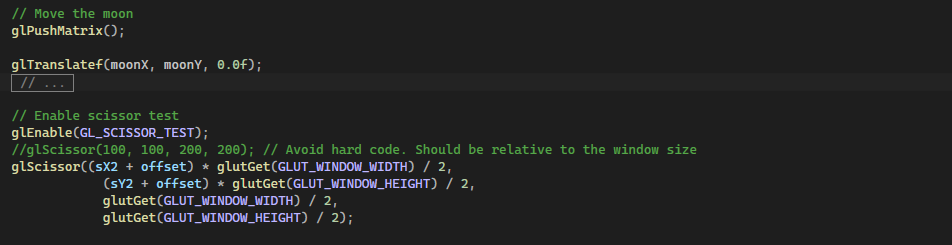
And it will look like this:



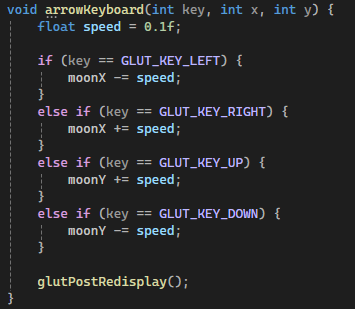
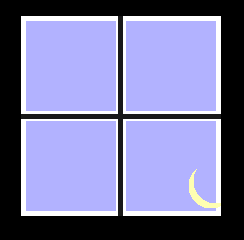
1. **Mask on Window & Move the Moon**

I was just thinking maybe if I could apply the masking on the window, it would be cool too. First step I do is creating the window buffer to clip the moon and only show on some defined area. Using glscissor, I need to set the origin x and y, and also the size of the clipping window in pixels unit. Therefore, I need to convert the window size to make the clipping window relative.

To move the moon, I use gltranslatef function where the new position is simply obtained from the delta position defined in arrwkeyboard function. This function simply add or subs the x and y of the moon.

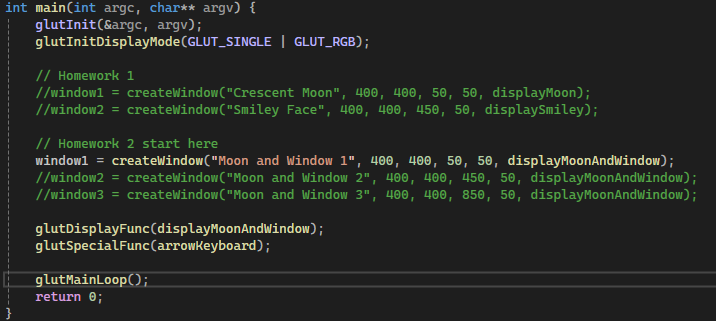


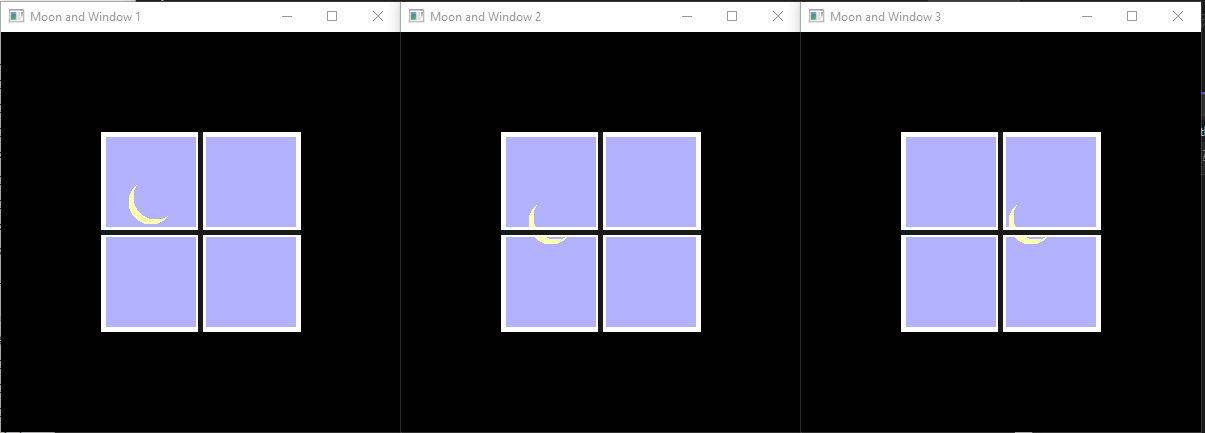


1. **Results**

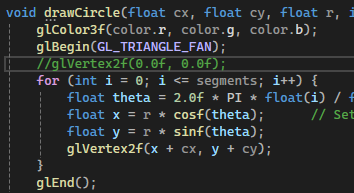
The last step is just do the functions callback at main function. Honestly, I just modify the previous homework, so that I don’t need to redo some functions.



****

Notes:

* I push my code here: <https://github.com/ardiawanbagusharisa/cgopengl>
* I just realized that I write unnecessary code, after I finish my code, and then re-evaluate using Sohan’s.



The complete code:

#include <GL/glew.h>

#include <GL/freeglut.h>

#include <cmath>

const float PI = 3.14159265359f;

struct Color {

float r, g, b;

};

float moonX = 0.0f, moonY = 0.0f; // Moon's position

int window1, window2, window3;

void drawRect(float cx, float cy, float sizeX, float sizeY, Color color) {

glColor3f(color.r, color.g, color.b);

glBegin(GL\_QUADS); // Start drawing a square

glVertex2f(cx - sizeX / 2, cy - sizeY / 2); // Change the method to be more parametric

glVertex2f(cx + sizeX / 2, cy - sizeY / 2);

glVertex2f(cx + sizeX / 2, cy + sizeY / 2);

glVertex2f(cx - sizeX / 2, cy + sizeY / 2);

glEnd();

}

void drawCircle(float cx, float cy, float r, int segments, Color color) {

glColor3f(color.r, color.g, color.b);

glBegin(GL\_TRIANGLE\_FAN);

for (int i = 0; i <= segments; i++) {

float theta = 2.0f \* PI \* float(i) / float(segments); // Compute the radian angle

float x = r \* cosf(theta); // Set coordinates of points on the perimeter of the circle using polar to cartesian

float y = r \* sinf(theta);

glVertex2f(x + cx, y + cy);

}

glEnd();

}

int createWindow(const char\* title, int width, int height, int posX, int posY, void (\*displayFunc)()) {

glutInitWindowSize(width, height);

glutInitWindowPosition(posX, posY);

int windowID = glutCreateWindow(title);

glewInit();

glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black background

glutDisplayFunc(displayFunc); // Register the display function

return windowID;

}

void displayMoon() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawCircle(0.0f, 0.0f, 0.5f, 100, { 1.0f, 1.0f, 0.7f }); // First circle, a little bit yellow

drawCircle(0.2f, 0.1f, 0.5f, 100, { 0.0f, 0.0f, 0.0f }); // Slightly moved black circle

glFlush();

}

void displaySmiley() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawCircle(0.0f, 0.0f, 0.6f, 100, { 1.0f, 0.8f, 0.6f }); // Main face

drawCircle(-0.275f, 0.275f, 0.2f, 100, { 1.0f, 1.0f, 1.0f }); // Left eye

drawCircle(-0.25f, 0.25f, 0.1f, 100, { 0.0f, 0.0f, 0.0f });

drawCircle(0.275f, 0.275f, 0.2f, 100, { 1.0f, 1.0f, 1.0f }); // Right eye

drawCircle(0.25f, 0.25f, 0.1f, 100, { 0.0f, 0.0f, 0.0f });

drawCircle(0.0f, -0.25f, 0.2f, 100, { 1.0f, 0.0f, 0.0f }); // Mouth

drawRect(0.0f, -0.15f, 0.4f, 0.2f, { 1.0f, 0.8f, 0.6f });

drawCircle(-0.35f, 0.0f, 0.175f, 100, { 1.0f, 0.7f, 0.5f }); // Cheeks

drawCircle(0.35f, 0.0f, 0.175f, 100, { 1.0f, 0.7f, 0.5f });

drawCircle(0.0f, 0.0f, 0.2f, 100, { 1.0f, 0.6f, 0.4f }); // Nose

drawCircle(-0.1f, 0.1f, 0.05f, 100, { 1.0f, 1.0f, 1.0f }); // Highlight

glFlush();

}

void displayMoonAndWindow() {

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_STENCIL\_BUFFER\_BIT);

// Draw main window

float cX = 0.0f, cY = 0.0f;

float sX = 1.0f, sY = 1.0f;

drawRect(cX, cY, sX, sY, { 1.0f, 1.0f, 1.0f });

// Draw the four smaller windows with relative size

float offset = 0.05f;

float sX2 = (sX / 2) - offset, sY2 = (sX / 2) - offset;

drawRect(cX + sX2 / 2 + offset / 2, cY + sY2 / 2 + offset / 2, sX2, sY2, { 0.7f, 0.7f, 1.0f });

drawRect(cX - sX2 / 2 - offset / 2, cY + sY2 / 2 + offset / 2, sX2, sY2, { 0.7f, 0.7f, 1.0f });

drawRect(cX - sX2 / 2 - offset / 2, cY - sY2 / 2 - offset / 2, sX2, sY2, { 0.7f, 0.7f, 1.0f });

drawRect(cX + sX2 / 2 + offset / 2, cY - sY2 / 2 - offset / 2, sX2, sY2, { 0.7f, 0.7f, 1.0f });

// Move the moon

glPushMatrix();

glTranslatef(moonX, moonY, 0.0f);

// Deleted. Use stencil buffer instead.

//drawCircle(cX + sX2 / 2 + offset / 2, cY + sY2 / 2 + offset / 2, sX2 / 4, 100, { 1.0f, 1.0f, 0.7f });

//drawCircle(cX + sX2 / 2 + offset, cY + sY2 / 2 + offset, sX2 / 4, 100, { 0.7f, 0.7f, 1.0f });

// Enable scissor test

glEnable(GL\_SCISSOR\_TEST);

//glScissor(100, 100, 200, 200); // Avoid hard code. Should be relative to the window size

glScissor((sX2 + offset) \* glutGet(GLUT\_WINDOW\_WIDTH) / 2,

(sY2 + offset) \* glutGet(GLUT\_WINDOW\_HEIGHT) / 2,

glutGet(GLUT\_WINDOW\_WIDTH) / 2,

glutGet(GLUT\_WINDOW\_HEIGHT) / 2);

// Draw the moon with mask method instead of two circles stacking

// Enable stencil testing for masking

glEnable(GL\_STENCIL\_TEST);

// 1. Draw the a circle into the stencil buffer

glStencilFunc(GL\_ALWAYS, 1, 0xFF); // Always write 1

glStencilOp(GL\_KEEP, GL\_KEEP, GL\_REPLACE);

glColorMask(GL\_FALSE, GL\_FALSE, GL\_FALSE, GL\_FALSE); // Disable color output

glDepthMask(GL\_FALSE); // Disable depth writing

drawCircle(cX + sX2 / 2 + offset / 2, cY + sY2 / 2 + offset / 2, sX2 / 4, 100, { 1.0f, 1.0f, 0.7f }); // Full moon

// 2. Subtract the second circle from stencil buffer

glStencilFunc(GL\_ALWAYS, 0, 0xFF); // Write 0 where we draw

glStencilOp(GL\_KEEP, GL\_KEEP, GL\_ZERO);

drawCircle(cX + sX2 / 2 + offset, cY + sY2 / 2 + offset, sX2 / 4, 100, { 0.7f, 0.7f, 1.0f }); // Masking circle

// 3. Draw only the remaining part of the stencil buffer

glColorMask(GL\_TRUE, GL\_TRUE, GL\_TRUE, GL\_TRUE); // Enable color output

glDepthMask(GL\_TRUE);

glStencilFunc(GL\_EQUAL, 1, 0xFF); // Draw only where stencil is 1

glStencilOp(GL\_KEEP, GL\_KEEP, GL\_KEEP);

drawCircle(cX + sX2 / 2 + offset / 2, cY + sY2 / 2 + offset / 2, sX2 / 4, 100, { 1.0f, 1.0f, 0.7f }); // Render the crescent

// Disable stencil test

glDisable(GL\_STENCIL\_TEST);

glDisable(GL\_SCISSOR\_TEST);

glPopMatrix();

// Draw the frame

drawRect(cX, cY, offset, sY, { 1.0f, 1.0f, 1.0f }); // White

drawRect(cX, cY, sX, offset, { 1.0f, 1.0f, 1.0f });

drawRect(cX, cY, offset / 2, sY, { 0.1f, 0.1f, 0.1f }); // Black

drawRect(cX, cY, sX, offset / 2, { 0.1f, 0.1f, 0.1f });

glFlush();

}

void arrowKeyboard(int key, int x, int y) {

float speed = 0.1f;

if (key == GLUT\_KEY\_LEFT) {

moonX -= speed;

}

else if (key == GLUT\_KEY\_RIGHT) {

moonX += speed;

}

else if (key == GLUT\_KEY\_UP) {

moonY += speed;

}

else if (key == GLUT\_KEY\_DOWN) {

moonY -= speed;

}

glutPostRedisplay();

}

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

// Homework 1

//window1 = createWindow("Crescent Moon", 400, 400, 50, 50, displayMoon);

//window2 = createWindow("Smiley Face", 400, 400, 450, 50, displaySmiley);

// Homework 2 start here

window1 = createWindow("Moon and Window 1", 400, 400, 50, 50, displayMoonAndWindow);

//window2 = createWindow("Moon and Window 2", 400, 400, 450, 50, displayMoonAndWindow);

//window3 = createWindow("Moon and Window 3", 400, 400, 850, 50, displayMoonAndWindow);

glutDisplayFunc(displayMoonAndWindow);

glutSpecialFunc(arrowKeyboard);

glutMainLoop();

return 0;

}