/\*\*

\* Joshua Melink

\* CSE 274

\* 9/5/2012

\* HW01

\* Draws different objects to the screen by modifying the surface pixel array.

\*/

#include "cinder/app/AppBasic.h"

#include "cinder/gl/gl.h"

#include "cinder/gl/Texture.h"

#include "cinder/ImageIo.h"

#include "Resources.h"

#include "stdio.h"

#include "math.h"

using namespace ci;

using namespace ci::app;

using namespace std;

class HW01App : public AppBasic {

public:

void setup();

void mouseDown( MouseEvent event );

void update();

void draw();

void prepareSettings(Settings\* settings);

private:

static const int kAppWidth=800;

static const int kAppHeight=600;

static const int surfaceSize=1024;

Surface\* mySurface;

uint8\_t\* pixelArray;

void clear(uint8\_t\* pixelArray);

void drawRectangle(uint8\_t\* pixelArray, int xCoord, int xSize, int yCoord, int ySize, Color8u blend);

void drawTriangle(uint8\_t\* pixelArray, int xCoord, int xSize, int yCoord, int ySize, Color8u blend);

void drawCircle(uint8\_t\* pixelArray, int xCenter, int yCenter, int radius, Color8u blend);

void gradient(uint8\_t\* pixelArray, int r1, int g1, int b1, int r2, int g2, int b2);

void blur(uint8\_t\* pixelArray);

};

// These variables are used in the update and mouseDown methods

int loopCounter = 0;

int flag = 0;

int clickCounter = 0;

void HW01App::prepareSettings(Settings\* settings)

{

(\*settings).setWindowSize(kAppWidth,kAppHeight);

(\*settings).setResizable(false);

}

/\*

\* Clears the given pixel array by turning the entire array black.

\* @param pixelArray: The pixel array to be modified.

\*/

void HW01App::clear(uint8\_t\* pixelArray)

{

Color8u blend = Color8u(0,0,0);

for (int y = 0; y < surfaceSize; y++)

{

for (int x = 0; x < surfaceSize; x++)

{

int pixel = 3 \* (x + y \* surfaceSize);

pixelArray[pixel] = blend.r;

pixelArray[pixel + 1] = blend.g;

pixelArray[pixel + 2] = blend.b;

}

}

}

/\*

\* Draws a rectangle on the screen.

\* @param pixelArray: The pixel array to be modified.

\* @param xCoord: The x-coordinate of the upper-left hand corner of the rectangle.

\* @param xSize: The width of the rectangle.

\* @param yCoord: The y-coordinate of the upper-left hand corner of the rectangle.

\* @param ySize: The height of the rectangle.

\* @param blend: The color to be assigned to the rectangle.

\*/

void HW01App::drawRectangle(uint8\_t\* pixelArray, int xCoord, int xSize, int yCoord, int ySize, Color8u blend)

{

for (int y = 0; y < surfaceSize; y++)

{

for (int x = 0; x < surfaceSize; x++)

{

if (x > xCoord && x < xCoord + xSize && y > yCoord && y < yCoord + ySize)

{

int pixel = 3 \* (x + y \* surfaceSize);

pixelArray[pixel] = blend.r;

pixelArray[pixel + 1] = blend.g;

pixelArray[pixel + 2] = blend.b;

}

}

}

}

/\*

\* Draws a triangle on the screen.

\* @param pixelArray: The pixel array to be modified.

\* @param xCoord: The x-coordinate of the upper-left hand corner of the triangle.

\* @param xSize: The width of the triangle.

\* @param yCoord: The y-coordinate of the upper-left hand corner of the triangle.

\* @param ySize: The height of the triangle.

\* @param blend: The color to be assigned to the triangle.

\*/

void HW01App::drawTriangle(uint8\_t\* pixelArray, int xCoord, int xSize, int yCoord, int ySize, Color8u blend)

{

int counter = 0;

for (int y = 0; y < surfaceSize; y++)

{

for (int x = 0; x < surfaceSize; x++)

{

if (x > xCoord && x <= 100 + xCoord + xSize - counter && y > yCoord && y <= 100 + yCoord + ySize)

{

int pixel = 3 \* (x + y \* surfaceSize);

pixelArray[pixel] = blend.r;

pixelArray[pixel + 1] = blend.g;

pixelArray[pixel + 2] = blend.b;

}

}

counter++;

}

}

/\*

\* Draws a circle on the screen.

\* @param pixelArray: The pixel array to be modified.

\* @param xCenter: The x-coordinate of the center point of the circle.

\* @param yCenter: The y-coordinate of the center point of the circle.

\* @param radius: The radius of the circle.

\* @param blend: The color to be assigned to the circle.

\*/

void HW01App::drawCircle(uint8\_t\* pixelArray, int xCenter, int yCenter, int radius, Color8u blend)

{

double squareRoot = 0;

for (int y = 0; y < surfaceSize; y++)

{

for (int x = 0; x < surfaceSize; x++)

{

if (pow((pow((double)yCenter - y, 2) + pow((double)xCenter - x, 2)), .5) <= radius)

{

int pixel = 3 \* (x + y \* surfaceSize);

pixelArray[pixel] = blend.r;

pixelArray[pixel + 1] = blend.g;

pixelArray[pixel + 2] = blend.b;

}

}

}

}

/\*

\* Shades the background one color at the top of the screen and slowly changes it until it becomes a completely

\* different color at the bottom of the screen.

\* @param pixelArray: The pixel array to be modified.

\* @param r1: The amount of red in the color at the top of the screen.

\* @param g1: The amount of green in the color at the top of the screen.

\* @param b1: The amount of blue in the color at the top of the screen.

\* @param r2: The amount of red in the color at the bottom of the screen.

\* @param g2: The amount of green in the color at the bottom of the screen.

\* @param b2: The amount of blue in the color at the bottom of the screen.

\*/

void HW01App::gradient(uint8\_t\* pixelArray, int r1, int g1, int b1, int r2, int g2, int b2)

{

double percent;

for (int y = 0; y < surfaceSize; y++)

{

for (int x = 0; x < surfaceSize; x++)

{

int pixel = 3 \* (x + y \* surfaceSize);

if (pixel >= 0 && pixel < surfaceSize \* surfaceSize \* 3)

{

percent = (double)y / surfaceSize;

pixelArray[pixel] = percent \* r2 + (1 - percent) \* r1;

pixelArray[pixel + 1] = percent \* g2 + (1 - percent) \* g1;

pixelArray[pixel + 2] = percent \* b2 + (1 - percent) \* b1;

}

}

}

}

/\*

\* Blurs each pixel with its surrounding 8 pixels to form an overall blurry image.

\* @param pixelArray: The pixel array to be modified.

\*/

void HW01App::blur(uint8\_t\* pixelArray)

{

int sumRed, sumGreen, sumBlue;

int averageRed, averageGreen, averageBlue;

for (int y = 1; y < surfaceSize - 1; y++)

{

for (int x = 1; x < surfaceSize - 1; x++)

{

sumRed = 0;

sumGreen = 0;

sumBlue = 0;

for (int averageY = -1; averageY <= 1; averageY++)

{

for (int averageX = -1; averageX <= 1; averageX++)

{

int pixel = 3 \* ((x + averageX) + (y + averageY) \* surfaceSize);

sumRed = sumRed + pixelArray[pixel];

sumGreen = sumGreen + pixelArray[pixel + 1];

sumBlue = sumBlue + pixelArray[pixel + 2];

}

}

int pixel = 3 \* (x + y \* surfaceSize);

averageRed = sumRed / 9;

averageGreen = sumGreen / 9;

averageBlue = sumBlue / 9;

pixelArray[pixel] = averageRed;

pixelArray[pixel + 1] = averageGreen;

pixelArray[pixel + 2] = averageBlue;

}

}

}

/\*

\* Establishes the background image.

\*/

void HW01App::setup()

{

mySurface = new Surface(surfaceSize, surfaceSize, false);

pixelArray = (\*mySurface).getData();

clear(pixelArray);

gradient(pixelArray, 180, 150, 90, 20, 255, 150);

drawRectangle(pixelArray, 300, 200, 400, 200, Color8u(255, 0, 0));

drawCircle(pixelArray, 400, 300, 100, Color8u(0, 0, 255));

drawCircle(pixelArray, 450, 250, 10, Color8u(0, 0, 0));

drawCircle(pixelArray, 475, 294, 5, Color8u(0, 0, 0));

drawCircle(pixelArray, 480, 297, 5, Color8u(0, 0, 0));

drawCircle(pixelArray, 485, 299, 5, Color8u(0, 0, 0));

drawCircle(pixelArray, 490, 300, 5, Color8u(0, 0, 0));

drawCircle(pixelArray, 495, 300, 5, Color8u(0, 0, 0));

drawCircle(pixelArray, 750, 50, 50, Color8u(255, 255, 0));

drawTriangle(pixelArray, 275, 200, 175, 200, Color8u(0, 255, 0));

blur(pixelArray);

}

/\*

\* Whenever the mouse is clicked on the sun, it changes to a moon and vice versa.

\* (essentially just changing colors).

\* @param event: The event we are concerned with in this method is the mouse click on a particular region

\* of the screen.

\*/

void HW01App::mouseDown( MouseEvent event )

{

int xPosition, yPosition;

xPosition = event.getX();

yPosition = event.getY();

if (xPosition > 700 && yPosition < 100 && clickCounter % 2 == 0)

{

drawCircle(pixelArray, 750, 50, 50, Color8u(255, 255, 255));

clickCounter++;

}

else if (xPosition > 700 && yPosition < 100 && clickCounter % 2 == 1)

{

drawCircle(pixelArray, 750, 50, 50, Color8u(255, 255, 0));

clickCounter++;

}

}

/\*

\* This method is called continuously to produce the effect of the eye opening and closing.

\*/

void HW01App::update()

{

pixelArray = (\*mySurface).getData();

drawCircle(pixelArray, 450, 230 + loopCounter, 10, Color8u(0, 0, 255));

if (loopCounter < 18 && flag == 0)

{

loopCounter++;

}

if (loopCounter == 18)

{

flag = 1;

}

if (flag == 1)

{

loopCounter = loopCounter - 3;

drawCircle(pixelArray, 450, 250, 10, Color8u(0, 0, 0));

}

if (loopCounter == 0)

{

flag = 0;

}

}

void HW01App::draw()

{

gl::draw(\*mySurface);

}

CINDER\_APP\_BASIC( HW01App, RendererGl )

