Data Structures and Object-Oriented Programming

SIMPLE_PAINTER

Files: mySystem_SimplePainter.h, mySystem_SimplePainter.cpp

A canvas is a two-dimensional surface for drawing. It is represented as a two-dimensional array of pixels. In other words, a pixel is an element of the canvas. We can use an object of the class vector3 to represent a pixel color. A color has three components: red, green, and blue. Their values are inside [0, 1]. For example:
vector3 color;
color.r : red
color.g : green
color.b : blue

The painter has a current color. Use the current color to draw.

Your program must not crash. Check carefully when the brush is at the boundary of the drawing canvas.

Table 3.1 Key usage

Key	Description
's', 'S'	Show the student information on the console window, including name, ID, and email address.
'i', 'I'	Show the title and key usage of this program.
'n', 'N'	Decrease the brush size.
'm', 'M'	Increase the brush size.
'1'	Set the brush color to red.
'2'	Set the brush color to green.
'3'	Set the brush color to blue.
'r'	Reset the canvas. Generate the red, green, and blue color components for each pixel randomly in a uniform manner.

```
/*
Randomly generate the colors of the bitmap.
*/
void SIMPLE_PAINTER::reset( )

/*
Get the canvas dimension.
nx : number of columns; ny : number of rows
*/
void SIMPLE_PAINTER::getCanvasDimension( int &nx, int &ny ) const

int SIMPLE_PAINTER::computeCanvasIndex( int x, int y, int nx, int ny )
```

```
/*
Get the color at pixel (x,y).
Store it to color.
*/
void SIMPLE_PAINTER::getColorAtPixel( int x, int y, vector3 &color ) const

/*
Show system title.
Show key usage.

Show all the key usage as shown in Table 3.1.
*/
void SIMPLE_PAINTER::askForInput( )
{
        // show the meaningful messages
        "SIMPLE_PAINTER::askForInput( )"
        "Please use keyboard and mouse to control"
        "i, I: …
        "s, S: …
        "1: …
        …
}
/*
Return true if the key event is handled.
Return false otherwise.

Make sure that the brush size is not larger than mMaxBrushSize.
*/
bool SIMPLE_PAINTER::handleKeyPressedEvent( int key )

// Return the brush size. Return mBrushSize
int SIMPLE_PAINTER::getBrushSize( ) const

//Set the current color
void SIMPLE_PAINTER::setColor( const vector3 &color )

/*
Set transparency
*/
void SIMPLE_PAINTER::setTransparency( float v )
```

```
/*
Use the brush to draw. The center is at (x,y). A disk is defined based on position (x,y) and
brush radius.
The radius of  the disk is set to ~2*mBrushSize.

Algorithm:

For each pixel in the disk centered at (x,y)
  do
    compute color and transparency
    draw at the pixel
*/
void SIMPLE_PAINTER::clickAt(double x, double y)
{
   vector3 color = mColor;

   int ref_s = mBrushSize/2;
   double S2 = ref_s*ref_s;
   int s = ref_s * 4;
   for ( int j = -s; j <= s; ++j ) {
      for ( int i = -s; i <= s; ++i ) {
         double rx = i;
         double ry = j;
         double w = 1.0; // weight
         double R2 = rx*rx + ry*ry;

         if ( R2 > 4.0*S2 ) continue; // this pixel is too far. ignore it
         if ( R2 > S2/2.0 ) {    // pixel is a bit far but not too far
            double k = S2/2.0 - R2;
            double d = 0.003;   // magic number
            w = pow(2.718281828, d*k ); // use the guassian function to compute weight w
         } else {
            w = 1.0; // pixel is near to the center (x,y). Set weight to a high value; here it's one.
         }
         //
         drawAt(x+i, y+j, w*color, w*mTransparency);
      }
   }
}

/*
Draw one pixel at (x,y) with color and weight w. If (x,y) is not inside the canvas, return.
Make sure that each color component is inside [0,1].
new color = current_color*(1.0-w*w) + w*w*color;
*/
void SIMPLE_PAINTER::drawAt(int x, int y, const vector3 &color, double w )
```